# پروژه اول: فریبرز کوئری لنگویج!

فریبرز که به تازگی با دستورات و روش های پردازش متن در جاوا آشنا شده فکر ساختن محلی برای ذخیره داده های برنامه هایش به سرش میزند تا در آینده از این پایگاه داده در پروژه هایش استفاده کند. اما از آنجا که فریبرز در جاوا تازه کار است، از شما که انرژی زیادی دارید :) درخواست کمک میکند.

پایگاه داده مدنظر او برای ذخیره و مدیریت داده‌ها استفاده می‌شود و از مدل سازی خاصی برای ذخیره اطلاعات استفاده می‌کند. در این پایگاه داده، اطلاعات به صورت سطر های یک یا چند جدول ذخیره می‌شوند که هر ستون جدول مربوط به یک ویژگی از سطر های جدول است.

در این پروژه هدف شبیه سازی این پایگاه داده و پیاده سازی برنامه ای است که با دریافت دستورات ذکر شده، تغییرات لازم را داده ها ایجاد کند و خروجی های مد نظر را چاپ کند.

## مدل سازی داده ها

فریبرز برای ذخیره سازی اطلاعات از مدل سازی جدولی استفاده میکند.

در این پایگاه داده، اطلاعات به صورت جداول ذخیره می‌شوند و هر جدول شامل سطرهای مختلفی است که هر سطر اطلاعات مربوط (رکورد) به یک مورد خاص را نمایش می‌دهد.

هر ردیف در جدول معادل با یک رکورد در مجموعه داده است هر ستون مشخص کننده‌ی یک ویژگی یا ویژگی‌هایی از آن رکورد است.

به عنوان مثال برای دانشجویان تازه ورود، می‌توان یک جدول در پایگاه داده فریبرز طراحی کرد که شامل اطلاعاتی مانند نام، نام خانوادگی، شماره دانشجویی، رشته تحصیلی، مقطع تحصیلی، وضعیت تحصیلی و سایر جزئیات مربوط به آنان باشد. به عنوان مثال، فرض کنید که اطلاعات دانشجویان در یک دانشگاه مورد بررسی قرار گیرد. یک جدول در پایگاه داده فریبرز برای این موضوع ممکن است به صورت زیر باشد:

| نام | نام خانوادگی | شماره دانشجویی | رشته تحصیلی | مقطع تحصیلی | وضعیت تحصیلی |
|---|---|---|---|---|---|
| آرمین | احمدی | 981234567 | مهندسی کامپیوتر | کارشناسی | در حال تحصیل |
| مریم | رضایی | 981234568 | مهندسی برق | کارشناسی | فارغ التحصیل |
| علی | کریمی | 981234569 | علوم ریاضی | کارشناسی | در حال تحصیل |
| فاطمه | محمدی | 981234570 | مهندسی مکانیک | کارشناسی | در حال تحصیل |

در این جدول، هر ستون نشان‌دهنده‌ی یک ویژگی از دانشجویان است و هر ردیف نشان‌دهنده‌ی یک دانشجو است. این اطلاعات می‌تواند برای مدیریت و پردازش اطلاعات مربوط به دانشجوهای تازه ورود به دانشگاه استفاده شود، از جمله ثبت نام، مدیریت شهریه، برنامه‌ریزی درسی و موارد مشابه.

## شیوه دریافت ورودی و خروجی

برنامه باید به صورت تعاملی[1] عمل کند. زمانی که برنامه اجرا می شود، منتظر دریافت اولین دستور از کاربر می ماند. پس از ورود اولین دستور، برنامه آن را پردازش کرده و خروجی اجرای دستور را چاپ میکند. مجدد برنامه منتظر دریافت دستور بعدی می شود و این چرخه تا دریافت کلید واژه خاصی (مثلا quit) ادامه پیدا میکند.

برای چاپ سطر های خروجی، فرمت چاپ آزاد و به عهده خودتان است، ولی دقت داشته باشید که نام ستون ها و همچنین مقدار متناظر با هر ستون باید به خوبی قابل تشخیص باشد.

برای دریافت دستورات ورودی به نکات زیر دقت کنید:

- تمام دستور در یک سطر دریافت و پردازش میشود و خروجی متناظر با آن چاپ میشود.
- بین توکن های دستور (نام، پرانتز، کاما و ...) میتواند تعداد دلخواهی فاصله (space) قرار بگیرد و در اجرای دستور تاثیری ندارد.
- ورودی حساس به حروف بزرگ و کوچک (case-sensitive) نیست.

## مدیریت خطا ها

یکی از مهم ترین ویژگی های یک نرم افزار خوب! برخورد مناسب با خطا هاست. بنابراین برنامه باید قادر باشد به صورت مناسب با حالت های خطا برخورد کند. به عنوان مثال در صورتی که کاربر دستوری تعریف نشده وارد کند، برنامه نباید متوقف شود و باید با خطای مناسب کاربر را از مشکل مطلع کند.

در ادامه به ازای هر دستور حالت های خطایی که برنامه شما ممکن است با آنها مواجه شود ذکر شده، حین پیاده سازی توجه لازم را به این حالت های خطا داشته باشید و در صورت برخورد با آن ها، با متن خطای مناسب، کاربر را از مشکل مطلع کنید.

قالب کلی دستورات این برنامه به صورت زیر میباشد:

```
command table(parameter1, …)[argument1, …]{variable1, …}
```

⚠ **توجه:** در صورتی که هر کدام از قسمت های ذکر شده خالی باشد (دستور پارامتر، آرگومان یا متغیر نداشته باشد) نیازی به ذکر کردن نماد آن ها نیست و آن قسمت حذف میشود.

مثال: دستور بدون آرگومان:

```
command table(parameter1, …){variable1, …}
```

❌ **خطاهای کلی:** قالب دستور مطابق ساختار ذکر شده رعایت نشده باشد - دستور موجود نباشد.

در ادامه دستورات مختلف به همراه عملکرد آنها ذکر شده است:

- **دستور create:** از این دستور برای ایجاد یک جدول جدید استفاده میشود. اطلاعات ستون های جدول شامل نام و نوع آنها به عنوان *آرگومان* به دستور داده میشوند. همچنین این دستور خروجی ندارد.

🧩 **مثال:** ایجاد جدول با نام `students` و ستون های `id` و `name` به ترتیب برای شماره دانشجویی و نام دانشجو، همچنین ستون های `grade` و `approved` برای نمره و همچنین تعیین این موضوع که آیا ثبت نام دانشجو در درس تایید شده است (مقدار ۱) یا خیر (مقدار ۰) :

```
create students[id int, name str, grade dbl, approved int]
```

نوع داده های برنامه محدود به موارد زیر میباشد:

| نوع داده | توضیحات | مثال | مقدار پیشفرض | معادل جاوا |
|---|---|---|---|---|
| str | نوع داده رشته متنی | 'AP is Fun!' | '' | String |
| int | نوع داده عدد صحیح | 5920 | 0 | Integer |
| dbl | نوع داده عدد اعشاری | 5.219 | 0.0 | Double |

نام ستون ها و جدول باید فقط شامل حروف الفبا، اعداد و کاراکتر زیر خط (_) باشد.

❌ **خطاها:** نام جدول از قبل موجود باشد - لیست ستون ها خالی باشد - نوع داده نامعتبر باشد.

- **دستور drop:** این دستور تنها نام جدول را گرفته و آن را به همراه تمام اطلاعات حذف میکند و خروجی ندارد.

🧩 **مثال:** حذف جدول `students`:

```
drop students
```

❌ **خطاها:** جدول از قبل وجود نداشته باشد.

- **دستور add:** با استفاده از این دستور میتوان یک سطر به یک جدول اضافه کرد. اطلاعات سطر به عنوان متغیر به دستور داده میشوند. در خروجی این دستور، اطلاعات سطر اضافه شده را برمیگرداند.

🧩 **مثال:** اضافه کردن سطر به جدول `students`:

```
add students{id=40211343, name='ali'}
```

در صورتی که ستونی مقدار دهی نشود، مقدار پیش فرض آن نوع داده را دریافت میکند.

❌ **خطاها:** جدول از قبل وجود نداشته باشد - نام ستون برای مقدار دهی وجود نداشته باشد - برای مقدار دهی یک ستون از داده نامعتبر استفاده شود.

- **دستور get:** سطر های یک جدول را (بر اساس فیلتر داده شده) برمیگرداند. اطلاعات فیلتر به صورت *پارامتر* برای دستور ارسال میشود.

🧩 **مثال:** دریافت همه سطر های جدول `students`:

```
get students
```

🧩 **مثال:** دریافت همه سطر های با `grade` بیشتر از 15 در جدول `students`:

```
get students(grade > 15.0)
```

عملگر های مقایسه برای فیلتر، محدود به موارد زیر میباشد:

| عملگر | توضیحات | مثال |
| --- | --- | --- |
| = | بررسی تساوی | name = 'ali' |
| <, > | بررسی کوچکتر یا بزرگتر | grade < 10.0 |
| +, - | عملیات جمع یا تفاضل | last_grade < 20 - grade + 10 |

دقت کنید که طرفین عملگر های مقایسه ای میتواند هر ترکیب معتبر دلخواه از اعداد، ستون ها و عملگر های جمع و تفریق باشد.

❌ **خطاها:** جدول از قبل وجود نداشته باشد - نام ستون برای فیلتر وجود نداشته باشد - برای فیلتر یک ستون از نوع داده نامعتبر استفاده شود.

**دستور set:** این دستور اطلاعات یک (یا چند) سطر از جدول را (بر اساس فیلتر داده شده) بروزرسانی میکند. اطلاعات فیلتر به صورت *پارامتر* و مقادیر جدید به صورت متغیر به دستور داده میشود. در خروجی تعداد سطر هایی که در نتیجه این دستور بروزرسانی شدند را برمیگرداند.

🛠 **مثال:** بروزرسانی همه سطر های جدول students:

```
set students{approved=1}
```

🛠 **مثال:** تغییر name و grade سطر با id=40211343 از جدول students:

```
set students(id=40211343){name='reza', grade=18.5}
```

❌ **خطاها:** مشابه خطاهای دستور get برای فیلتر + خطاهای دستور add برای بروزرسانی.

**دستور del:** سطر های جدول را (بر اساس فیلتر داده شده) حذف میکند. اطلاعات فیلتر به صورت *پارامتر* به دستور داده میشود. در خروجی تعداد سطر های حذف شده را برمیگرداند.

🛠 **مثال:** حذف همه سطر های جدول students:

```
del students
```

🛠 **مثال:** حذف سطر با id=40211343 از جدول students:

```
del students(id=40211343)
```

🧩 **مثال:** حذف سطر با `name=reza` و `grade=18.5` از جدول `students`:

```
del students(name='reza', grade=18.5)
```

❌ **خطاها:** مشابه خطاهای دستور get برای فیلتر.

🚨 **نکته پیاده سازی:** پیاده سازی شما باید مبتنی بر **متدهای استرینگ در جاوا باشد**. پیاده سازی برنامه مشابه زبان C نمره ای نخواهد داشت (مثلا در نظر گرفتن flag، یا استفاده از رشته به صورت آرایه ای کاراکترها).

💡 **راهنمایی:** برای پیاده سازی جداول میتوانید از یک آرایه دو بعدی از رشته ها، **`ArrayList`** و یا **`HashMap`** استفاده کنید. همچنین برای راحتی پیاده میتوانید فرض کنید ظرفیت هر جدول محدود به عددی خاص (به عنوان مثال ۱۰۰ سطر) می باشد، اما توجه کنید که این عدد باید به راحتی از داخل کد قابل تغیر باشد.