1) Scrivi una query che estragga il numero di persone con meno di 30 anni che guadagnano più di 50.000 dollari l'anno

```
SELECT count(*)
FROM records
WHERE over_50k = 1
AND age < 30
;
```

2) Scrivi una query che riporti il guadagno di capitale medio per ogni categoria lavorativa

```
SELECT workclasses.name AS workclass, avg(capital_gain)
FROM workclasses
JOIN records ON workclasses.id = records.workclass_id
GROUP BY workclasses.name
;
```

Si può usare LEFT JOIN in alternativa per ricavare anche le workclasses che non sono associate a nessun record.

3) Esercizio

Tecnologie utilizzate:

- Java 11;
- il server HTTP built-in in Java:
  - module: jdk.httpserver
  - package: com.sun.net.httpserver
- i driver sqlite-jdbc-3.34.0 per l'accesso al database (JAR presente nel repository):
  - https://github.com/xerial/sqlite-jdbchttps://github.com/xerial/sqlite-jdbc
- la libreria gson-2.8.6 per la serializzazione (JAR presente nel repository):
  - https://github.com/google/gsonhttps://github.com/google/gson
- l'IDE IntelliJ IDEA Community Edition.

I moduli di codice previsti sono:

- WebService:
  - inizializza il client per il database;
  - avvia il server assegnando a esso un WebHandler;
- WebHandler:
  - effettua il routing (manuale) delle richieste verso le funzioni controller;
  - contiene le funzioni di controller per ogni risorsa prevista;
- DatabaseClient:
  - mantiene la connessioni attiva al database;
  - compila ed esegue le query e restituisce al controller i dati strutturati in oggetti del linguaggio programmazione.

Interfaccia API (porta 8080 su localhost):

- record denormalizzati con paginazione parametrica:
  - GET "/records?offset=<int>&count=<int>"

- statistiche aggregate e filtrate;
  - GET “/stats?aggregationType=<string>&aggregationValue=<int>”
- record denormalizzati in csv (download in directory del repository):
  - GET “/csv”

Alcune cose da dire:
- vengono usati i PreparedStatement per evitare SQL Injection;
- non sono previsti Data Access Objects: i dati vengono restituiti dal DatabaseClient in strutture ArrayList o HashMap e gestiti dal controller in maniera agnostica rispetto allo schema dei dati.

Record denormalizzati:

```
SELECT
        records.id,
        records.age,
        workclasses.name AS workclass,
        education_levels.name AS education_level,
        records.education_num,
        marital_statuses.name AS marital_status,
        occupations.name AS occupation,
        races.name AS race,
        sexes.name AS sex,
        records.capital_gain,
        records.capital_loss,
        records.hours_week,
        countries.name AS country,
        records.over_50k
FROM records
JOIN workclasses ON workclasses.id = records.workclass_id
JOIN education_levels ON education_levels.id = records.education_level_id
JOIN marital_statuses ON marital_statuses.id = records.marital_status_id
JOIN occupations ON occupations.id = records.occupation_id
JOIN races ON races.id = records.race_id
JOIN sexes ON sexes.id = records.sex_id
JOIN countries ON countries.id = records.country_id
WHERE records.id > ?
LIMIT ?
;
```

Statistiche aggregate filtrate:

```
SELECT
        sum(capital_gain),
        avg(capital_gain),
        sum(capital_loss),
        avg(capital_loss),
        sum(case when over_50k = 1 then 1 else 0 end) AS over_50k,
        sum(case when over_50k = 0 then 1 else 0 end) AS under_50k
FROM records
```

```
WHERE <aggregationType> = ?
GROUP BY <aggregationType>
;
```