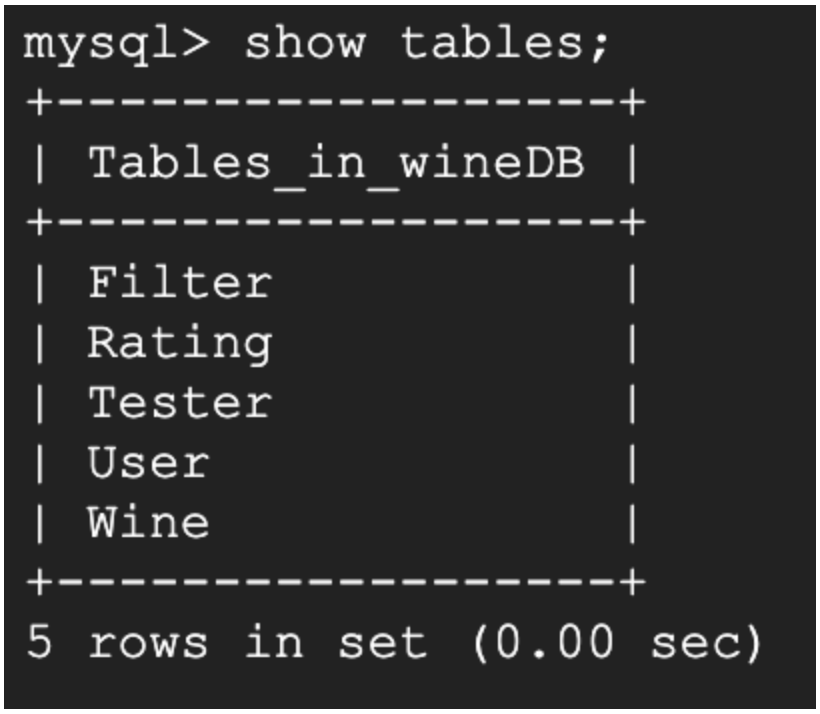
Stage 3 docs

1. **Implement at least 4 main tables**

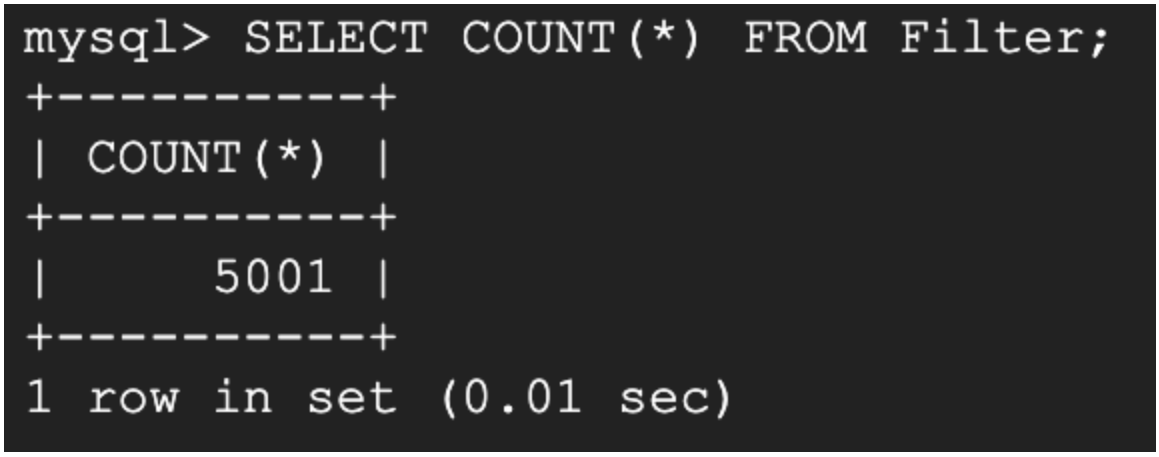


2. DDL commands that we used to create each tables

```
CREATE TABLE Wine (
  wineID INT NOT NULL,
  name VARCHAR(255),
  description VARCHAR(1500),
  winery VARCHAR(255),
  PRIMARY KEY(wineID)
);

CREATE TABLE User (
  userID INT NOT NULL,
  name VARCHAR(255),
  PRIMARY KEY(userID)
);

CREATE TABLE Tester (
  testerID INT NOT NULL,
```

```
  name VARCHAR(255),
  twitter VARCHAR(255),
  PRIMARY KEY(testerID)
);

CREATE TABLE Rating (
  ratingID INT NOT NULL,
  wineID INT,
  score DECIMAL,
  review VARCHAR(255),
  userID INT,
  PRIMARY KEY(ratingID),
  FOREIGN KEY(wineID) REFERENCES Wine(wineID),
  FOREIGN KEY(userID) REFERENCES User(userID)
);

CREATE TABLE Filter (
  filterID INT NOT NULL,
  wineID INT,
  country VARCHAR(255),
  points INT,
  price INT,
  variety VARCHAR(255),
  PRIMARY KEY(filterID),
  FOREIGN KEY(wineID) REFERENCES Wine(wineID)
);
```

3. **Insert data to tables. Insert at least 1000 rows in each table**



```
mysql> SELECT COUNT(*) FROM Rating;
+----------+
| COUNT(*) |
+----------+
|     5001 |
+----------+
1 row in set (0.01 sec)
```

```
mysql> SELECT COUNT(*) FROM Tester;
+----------+
| COUNT(*) |
+----------+
|     5001 |
+----------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM User;
+----------+
| COUNT(*) |
+----------+
|     5001 |
+----------+
1 row in set (0.01 sec)
```

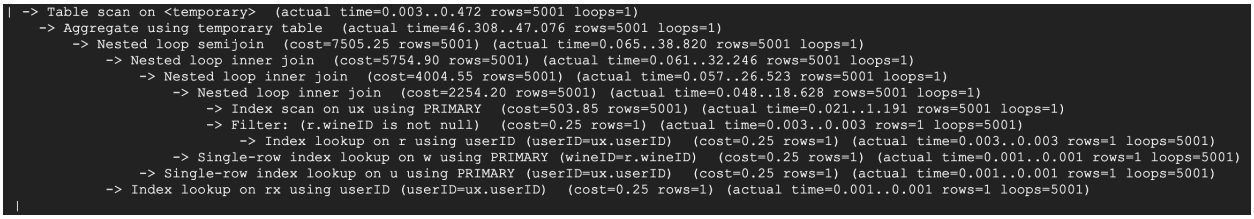
```
mysql> SELECT COUNT(*) FROM Wine;
+----------+
| COUNT(*) |
+----------+
|     5001 |
+----------+
1 row in set (0.00 sec)
```

4. **Advanced Queries**

Get Average Rating of Wine

```
SELECT w.name, ROUND(AVG(r.score),2) As AverageRating
FROM Wine w NATURAL JOIN Rating r JOIN User u on r.userID = u.userID
WHERE u.userID IN (
    SELECT ux.userID
    FROM Rating rx NATURAL JOIN User ux
    GROUP BY ux.userID
)
GROUP BY wineID
```

<Indexing report>



(Process flow of the query before indexing)

```
| -> Table scan on <temporary>  (actual time=0.003..0.552 rows=5001 loops=1)
    -> Aggregate using temporary table  (actual time=52.138..53.192 rows=5001 loops=1)
        -> Nested loop semijoin  (cost=7505.25 rows=5001) (actual time=0.093..43.140 rows=5001 loops=1)
            -> Nested loop inner join  (cost=5754.90 rows=5001) (actual time=0.086..35.990 rows=5001 loops=1)
                -> Nested loop inner join  (cost=4004.55 rows=5001) (actual time=0.081..29.965 rows=5001 loops=1)
                    -> Nested loop inner join  (cost=2254.20 rows=5001) (actual time=0.072..20.814 rows=5001 loops=1)
                        -> Index scan on ux using PRIMARY  (cost=503.85 rows=5001) (actual time=0.043..1.373 rows=5001 loops=1)
                        -> Filter: (r.wineID is not null)  (cost=0.25 rows=1) (actual time=0.003..0.004 rows=1 loops=5001)
                            -> Index lookup on r using userID (userID=ux.userID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=5001)
                    -> Single-row index lookup on w using PRIMARY (wineID=r.wineID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=5001)
                -> Single-row index lookup on u using PRIMARY (userID=ux.userID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5001)
            -> Index lookup on rx using userID (userID=ux.userID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5001)
|
```

(after indexing)

```
mysql> DROP INDEX rating_idx ON Rating;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

(dropped index due to inefficiency)

```
+-----------------------------------------------------------------------------------------+----------------+
| name                                                                                    | AverageRating |
+-----------------------------------------------------------------------------------------+----------------+
| name                                                                                    |          0.00 |
| Nicosia 2013 Vulkà Bianco (Etna)                                                        |          5.00 |
| Quinta dos Avidagos 2011 Avidagos Red (Douro)                                           |          5.00 |
| Rainstorm 2013 Pinot Gris (Willamette Valley)                                           |          3.00 |
| St. Julian 2013 Reserve Late Harvest Riesling (Lake Michigan Shore)                     |          4.00 |
| Sweet Cheeks 2012 Vintner's Reserve Wild Child Block Pinot Noir (Willamette Valley)     |          1.00 |
| Tandem 2011 Ars In Vitro Tempranillo-Merlot (Navarra)                                   |          4.00 |
| Terre di Giurfo 2013 Belsito Frappato (Vittoria)                                        |          2.00 |
| Trimbach 2012 Gewurztraminer (Alsace)                                                   |          5.00 |
| Heinz Eifel 2013 Shine Gewürztraminer (Rheinhessen)                                     |          3.00 |
| Jean-Baptiste Adam 2012 Les Natures Pinot Gris (Alsace)                                 |          1.00 |
| Kirkland Signature 2011 Mountain Cuvée Cabernet Sauvignon (Napa Valley)                 |          5.00 |
| Leon Beyer 2012 Gewurztraminer (Alsace)                                                 |          2.00 |
| Louis M. Martini 2012 Cabernet Sauvignon (Alexander Valley)                             |          0.00 |
| Masseria Setteporte 2012 Rosso (Etna)                                                   |          2.00 |
+-----------------------------------------------------------------------------------------+----------------+
```

For this query, the score in the Rating table was used for indexing, because it is the attribute that is accessed the most including when computing the average score. With the indexing, however, the cost and the runtime increased. Due to the inefficiency with the indexing, the index was dropped.

Acquire Cheap (less than 20 USD) but highly rated Wine (greater than or equal to 90 points and average scores of 4 or above)

SELECT w.name, f.price, sub.avgScore, f.points

```
FROM Wine w NATURAL JOIN Filter f JOIN (
    SELECT wineID, AVG(score) as avgScore
    FROM Rating
    GROUP BY wineID
) as sub ON w.wineID = sub.wineID
WHERE f.price<20 AND f.points >=90 AND sub.avgScore >= 4
```

<Indexing report>

```
| -> Nested loop inner join  (cost=279922.03 rows=2778333) (actual time=10.072..12.600 rows=70 loops=1)
    -> Nested loop inner join  (cost=699.54 rows=556) (actual time=0.151..2.414 rows=245 loops=1)
        -> Filter: ((f.price < 20) and (f.points >= 90) and (f.wineID is not null))  (cost=505.10 rows=556) (actual time=0.135..1.956 rows=245 loops=1)
            -> Table scan on f  (cost=505.10 rows=5001) (actual time=0.071..1.510 rows=5001 loops=1)
        -> Single-row index lookup on w using PRIMARY (wineID=f.wineID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=245)
    -> Index lookup on sub using <auto_key0> (wineID=f.wineID)  (actual time=0.001..0.001 rows=0 loops=245)
        -> Materialize  (cost=1504.30..1504.30 rows=5001) (actual time=10.126..10.140 rows=1617 loops=1)
            -> Filter: (avg(Rating.score) >= 4)  (cost=1004.20 rows=5001) (actual time=0.151..8.358 rows=1617 loops=1)
                -> Group aggregate: avg(Rating.score)  (cost=1004.20 rows=5001) (actual time=0.144..6.999 rows=5001 loops=1)
                    -> Index scan on Rating using wineID  (cost=504.10 rows=5001) (actual time=0.139..5.563 rows=5001 loops=1)
 |
```

(before indexing)

```
| -> Nested loop inner join  (cost=279922.03 rows=2778333) (actual time=10.563..13.412 rows=70 loops=1)
    -> Nested loop inner join  (cost=699.54 rows=556) (actual time=0.367..2.986 rows=245 loops=1)
        -> Filter: ((f.price < 20) and (f.points >= 90) and (f.wineID is not null))  (cost=505.10 rows=556) (actual time=0.337..2.390 rows=245 loops=1)
            -> Table scan on f  (cost=505.10 rows=5001) (actual time=0.063..1.932 rows=5001 loops=1)
        -> Single-row index lookup on w using PRIMARY (wineID=f.wineID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=245)
    -> Index lookup on sub using <auto_key0> (wineID=f.wineID)  (actual time=0.001..0.001 rows=0 loops=245)
        -> Materialize  (cost=1504.30..1504.30 rows=5001) (actual time=10.362..10.376 rows=1617 loops=1)
            -> Filter: (avg(Rating.score) >= 4)  (cost=1004.20 rows=5001) (actual time=0.189..8.638 rows=1617 loops=1)
                -> Group aggregate: avg(Rating.score)  (cost=1004.20 rows=5001) (actual time=0.180..7.614 rows=5001 loops=1)
                    -> Index scan on Rating using wineID  (cost=504.10 rows=5001) (actual time=0.168..6.213 rows=5001 loops=1)
```

(created index on rating.score. no difference in runtime.)

```
| -> Nested loop inner join  (cost=326688.24 rows=3243345) (actual time=10.640..13.111 rows=70 loops=1)
    -> Nested loop inner join  (cost=732.09 rows=649) (actual time=0.111..2.350 rows=245 loops=1)
        -> Filter: ((f.price < 20) and (f.points >= 90) and (f.wineID is not null))  (cost=505.10 rows=649) (actual time=0.097..1.906 rows=245 loops=1)
            -> Table scan on f  (cost=505.10 rows=5001) (actual time=0.046..1.454 rows=5001 loops=1)
        -> Single-row index lookup on w using PRIMARY (wineID=f.wineID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=245)
    -> Index lookup on sub using <auto_key0> (wineID=f.wineID)  (actual time=0.001..0.001 rows=0 loops=245)
        -> Materialize  (cost=1504.30..1504.30 rows=5001) (actual time=10.699..10.712 rows=1617 loops=1)
            -> Filter: (avg(Rating.score) >= 4)  (cost=1004.20 rows=5001) (actual time=0.224..8.958 rows=1617 loops=1)
                -> Group aggregate: avg(Rating.score)  (cost=1004.20 rows=5001) (actual time=0.218..7.849 rows=5001 loops=1)
                    -> Index scan on Rating using wineID  (cost=504.10 rows=5001) (actual time=0.211..6.274 rows=5001 loops=1)
 |
```

(created index on f.price, f.points. increased runtime. bad)

name	price	avgScore	points
Beaumont 2005 Hope Marguerite Chenin Blanc (Walker Bay)	0	5.0000	90
Carl Graff 2014 Graacher Himmelreich Spätlese Riesling (Mosel)	14	4.0000	91
Tenuta di Sesta 2011 Riserva (Brunello di Montalcino)	0	4.0000	90
Château Lafon-Rochet 2011 Saint-Estèphe	0	4.0000	92
Dionigi 2006 Sagrantino di Montefalco	0	5.0000	92
Patrick Javillier 2011 Les Tillets (Meursault)	0	4.0000	92
Deutz 2007 Blanc de Blancs Brut Chardonnay (Champagne)	0	5.0000	92
Rafael Cambra 2012 Soplo Garnacha Tintorera (Valencia)	15	4.0000	90
Château Sainte Marguerite 2014 Rosé (Côtes de Provence)	0	5.0000	91
Château Mayne Vieil 2014 Fronsac	17	4.0000	90
CVA Canicattì 2015 Aquilae Bio Grillo (Terre Siciliane)	0	4.0000	90
Baglio del Cristo di Campobello 2015 Laluci White (Sicilia)	0	5.0000	90
I Luoghi 2008 Campo al Fico (Bolgheri Superiore)	0	5.0000	90
Rutherford Ranch 2013 Chardonnay (Napa Valley)	19	4.0000	90
Château d'Or et de Gueules 2010 Les Cimels Red (Costières de Nîmes)	17	4.0000	90

For this query, the score in the Rating table was indexed first, because it is the attribute that is accessed the most including when computing the average score. With the indexing, however, the cost and the runtime did not change. Because there is no advantage in indexing with this attribute, the index was dropped. Then, the price and the points in the Filter table were indexed to see if it leads to high efficiency. However, the runtime and the cost increased with the index. The index was dropped for the reason.