**Assignment 3 – ISM 6218 Adv Databases**

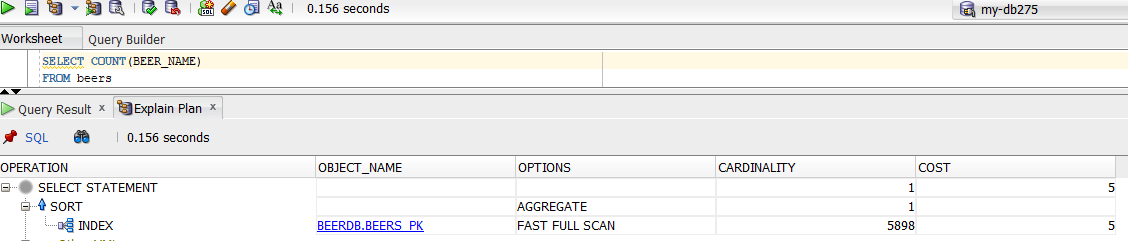
**GROUP 2: Alphonse Aloia, Shruthi Priya Athikam, Krishna Sai Surya Teja Basetty, Malavika Parakkat Byju, Sahil Shah**

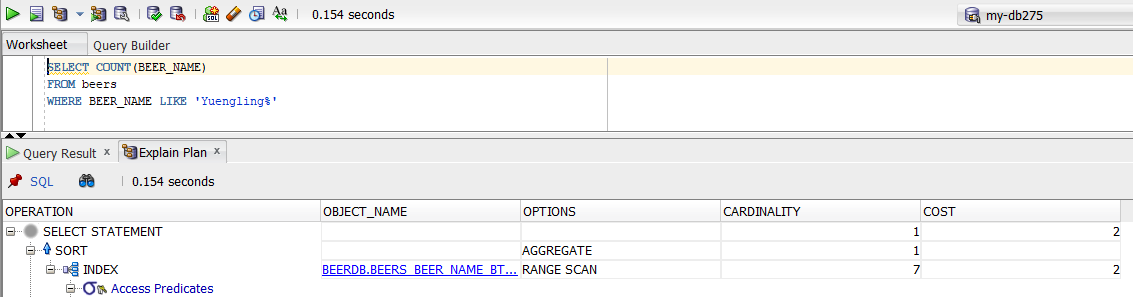
## **Idea 1: Investigating Selectivity**

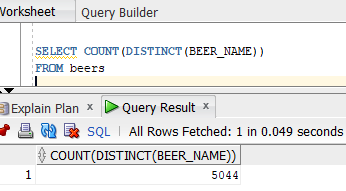
Look up selectivity in your database textbook.  Essentially, the optimizer makes a decision regarding index use based on the fraction of results returned. Indexes are most useful when selecting a small fraction of the available records.  You can conduct a simple experiment to find out where the cutoff percentage lies by developing a simple single table query and gradually shrinking (or growing) the query range.

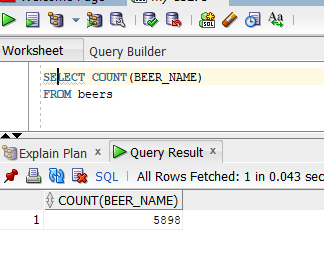
BEERS table in BEER DB

A number of queries were run to get a better insight on Selectivity using the beerdb.

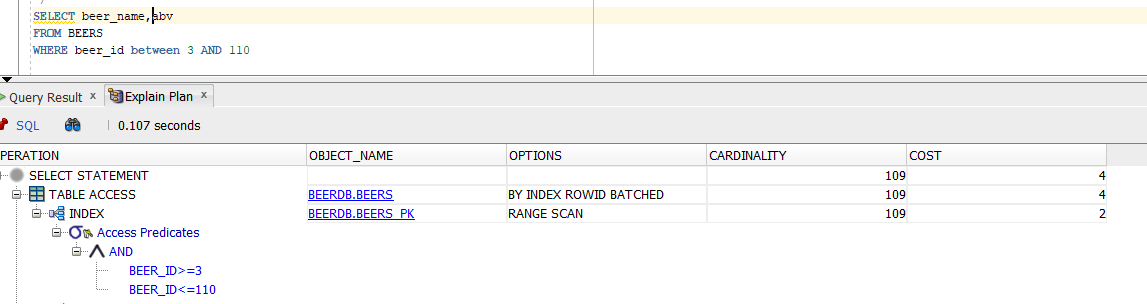


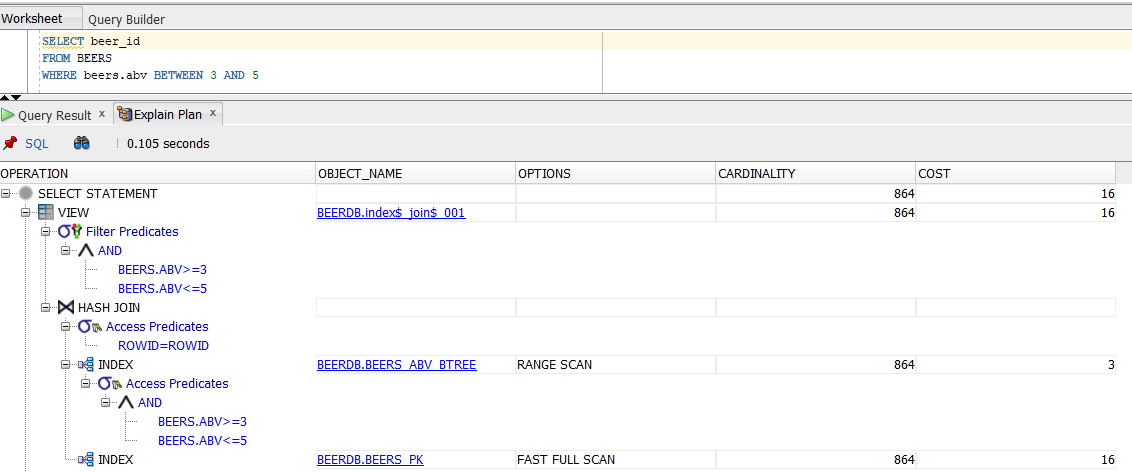






**Selectivity Ratio 0.855.**



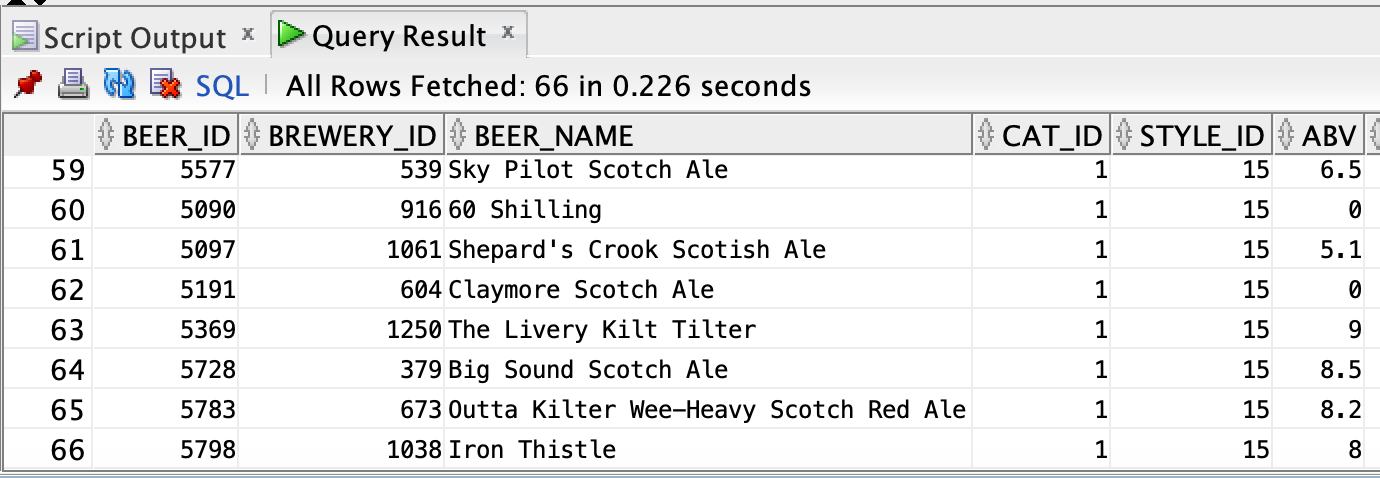


## **Idea 2: Start Simple and Show that Indexing Works**

Select \*

FROM BEER\_NAMES1

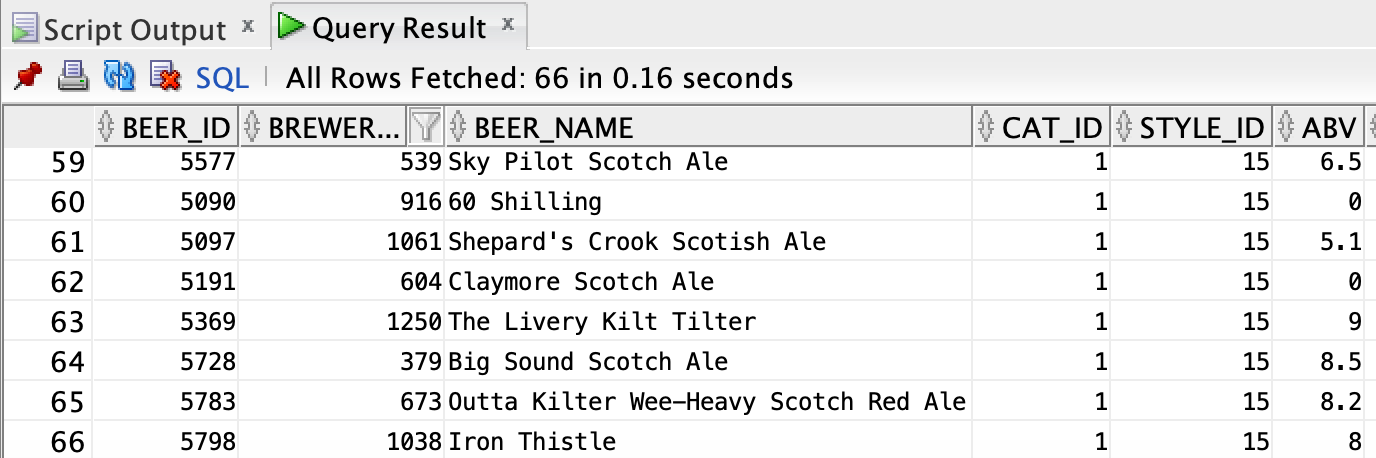
WHERE STYLE\_ID = 15;



*Figure 1 - Performance Before Indexing: 0.226 seconds*

CREATE INDEX style\_idx

ON BEER\_NAMES1(STYLE\_ID);



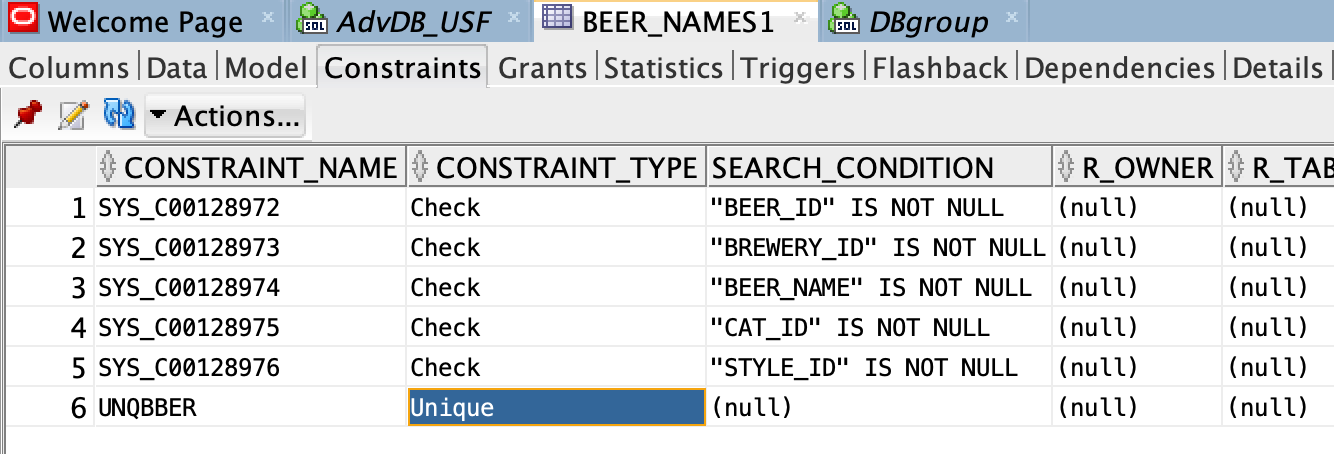
*Figure 2 - Performance After Indexing: 0.16 seconds*

**After indexing the fetching time reduced by 27.6%**

## **Idea 3: Primary Keys and Indexes**

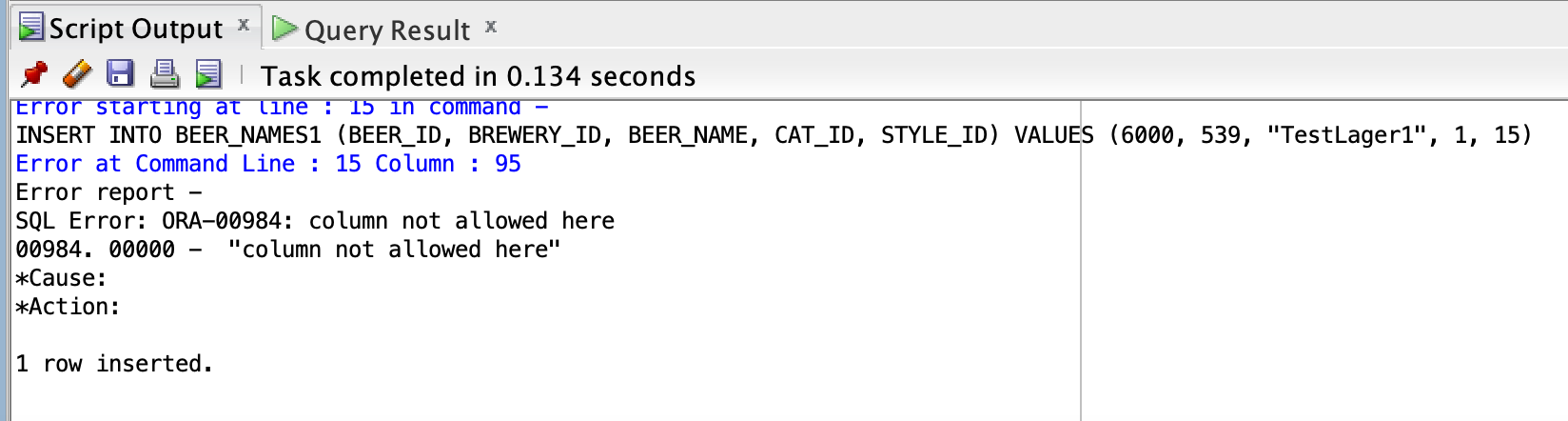
1. Adding a unique constraint on Beer\_ID

ALTER TABLE beer\_names1  
ADD CONSTRAINT unqbber UNIQUE (BEER\_ID);



1. Inserting new row of data

INSERT INTO BEER\_NAMES1 (BEER\_ID, BREWERY\_ID, BEER\_NAME, CAT\_ID, STYLE\_ID) VALUES (6000, 539, 'TestLager1', 1, 15);



*Figure 3 - Unique Constraint on Beer\_ID*

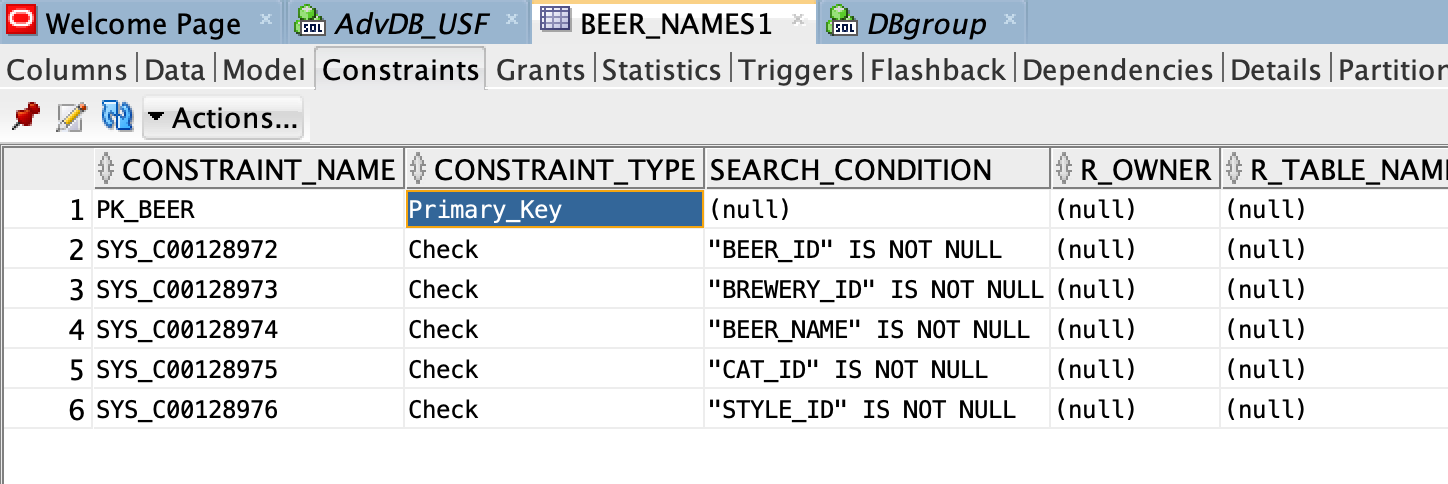
1. Performing same operation after adding Primary Key constraint on Beer\_ID

ALTER TABLE beer\_names1

DROP CONSTRAINT unqbber;

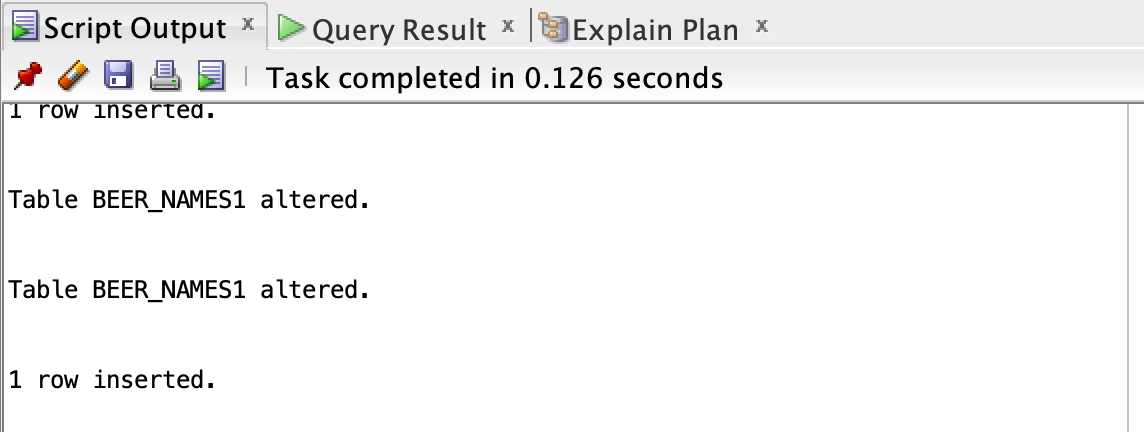
ALTER TABLE beer\_names1

ADD CONSTRAINT PK\_beer PRIMARY KEY (BEER\_ID);



*Figure 4 - Adding Primary Key*

INSERT INTO BEER\_NAMES1 (BEER\_ID, BREWERY\_ID, BEER\_NAME, CAT\_ID, STYLE\_ID) VALUES (6001, 539, 'TestLager2', 1, 15);



*Figure 5 - Output after adding PK Constraint*

Adding a primary key constraint reduced the time by 0.008s

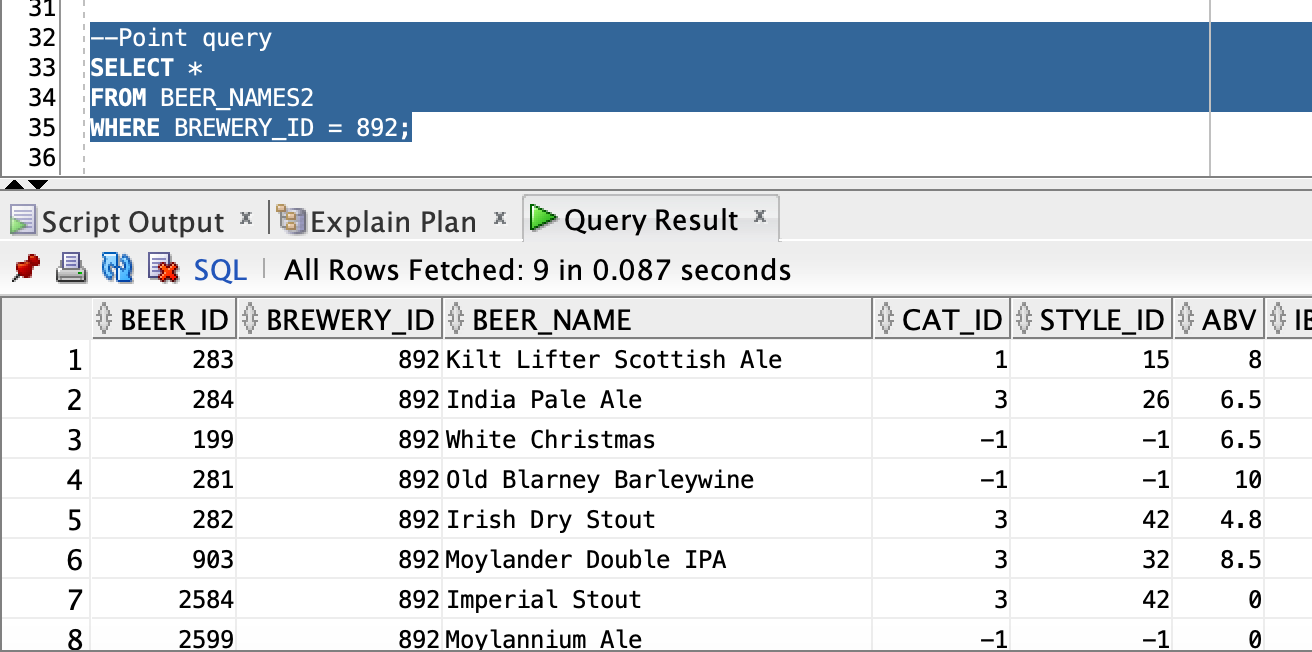
## **Idea 4: Indexing for Different Query Types**

CREATE TABLE BEER\_NAMES2 AS

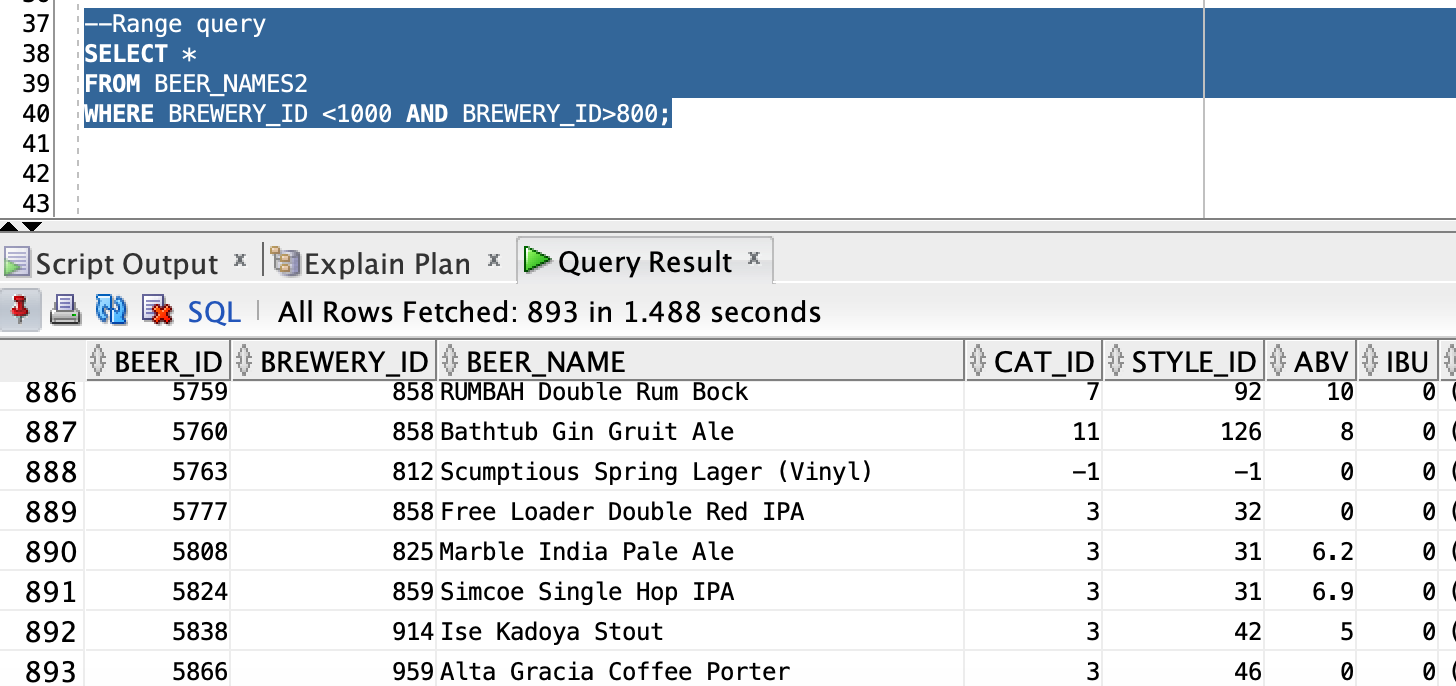
SELECT \* FROM BEERDB.beers;

**NON-INDEXED POINT, RANGE and SCAN QUERY results:**

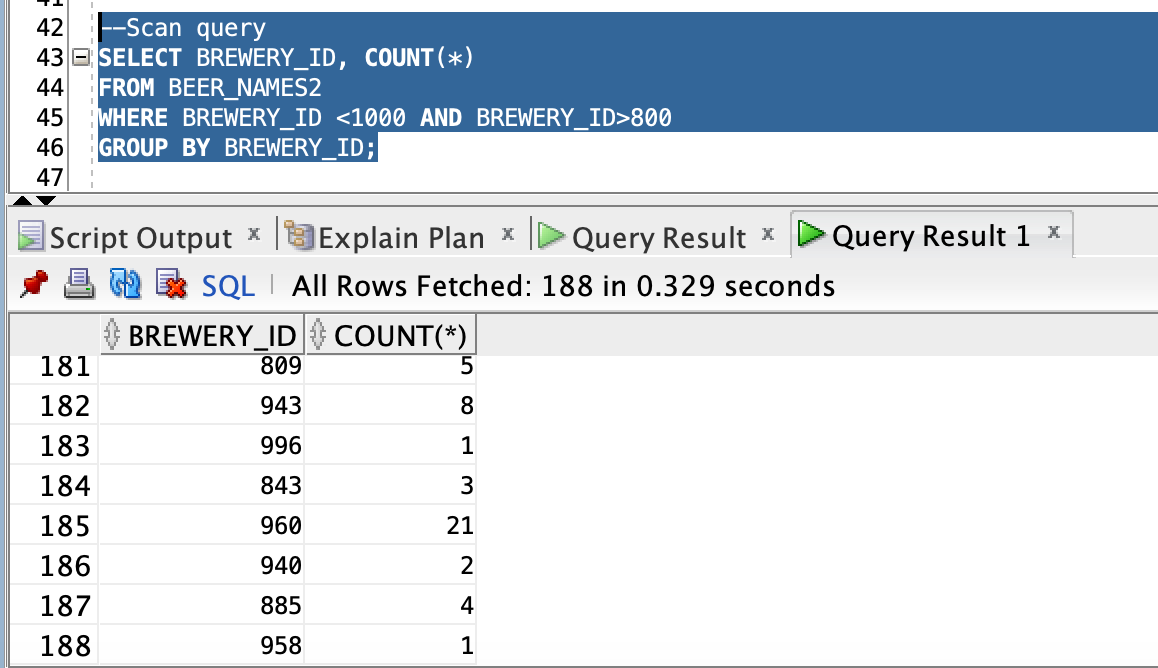
1. **Point Query**



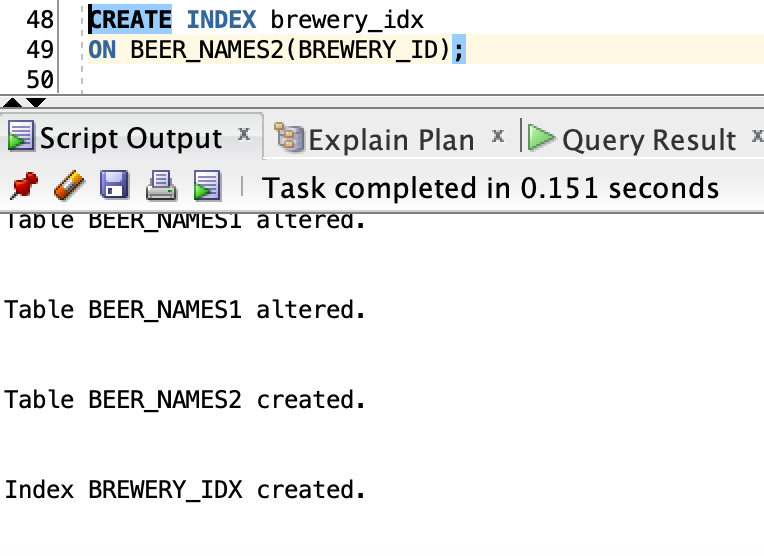
1. **Range Query**



1. **Scan Query**

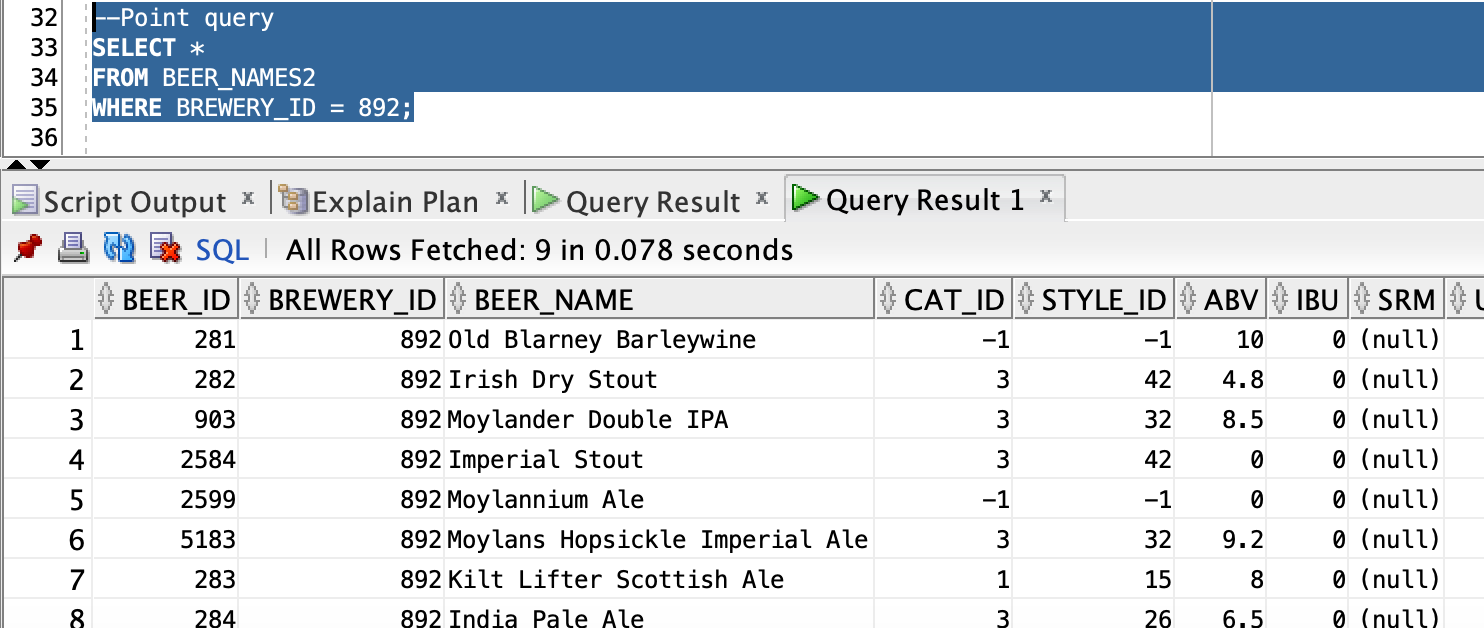


**Creating Index on Breweries\_ID**

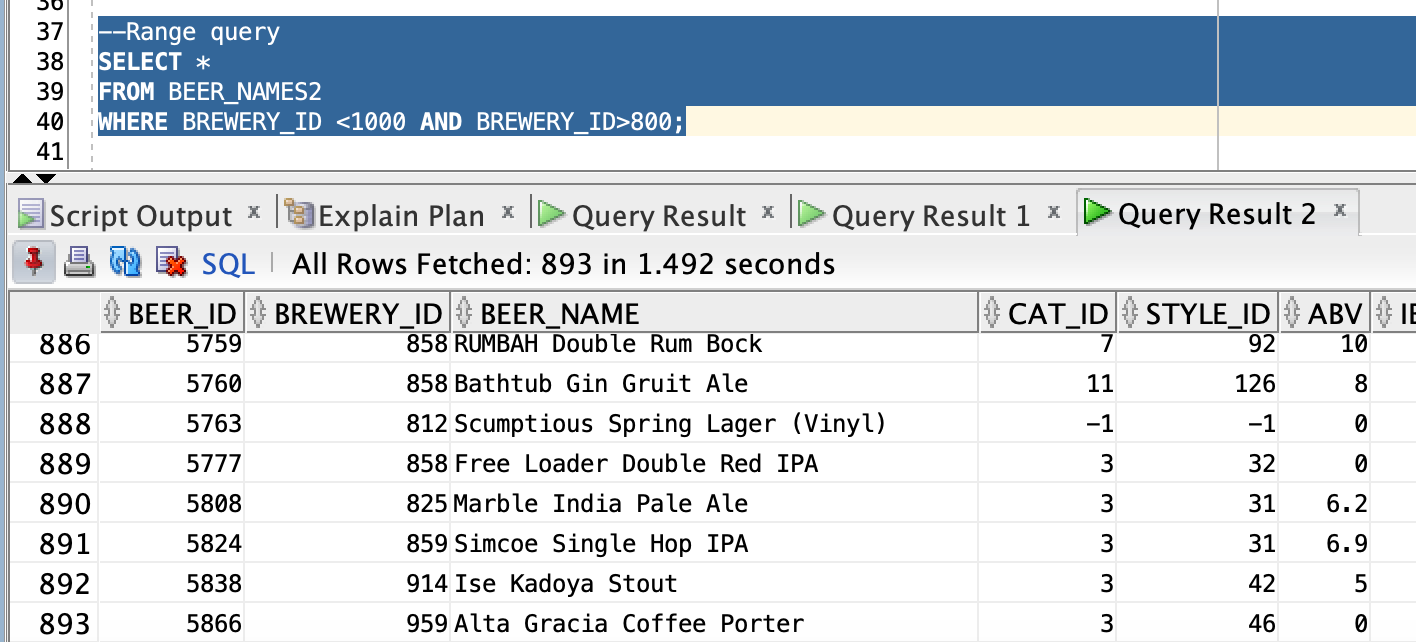


**INDEXED POINT, RANGE and SCAN QUERY RESULTS**

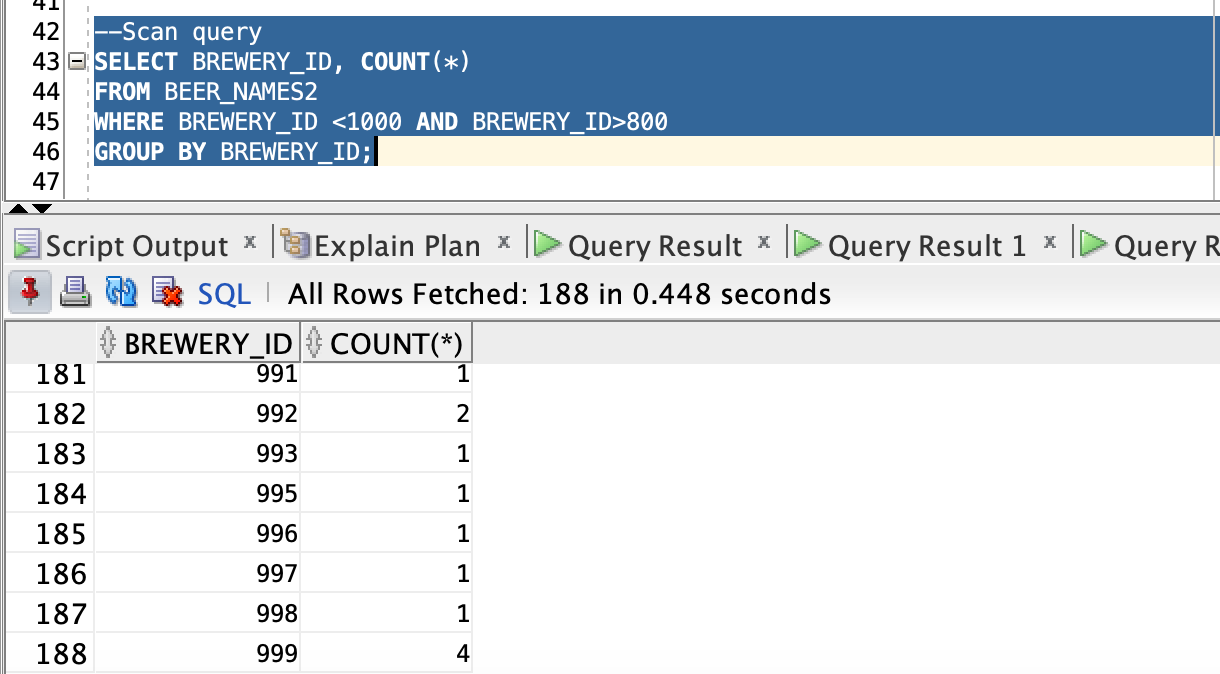
1. **Point Query**



1. **Range Query**



1. **Scan Query**

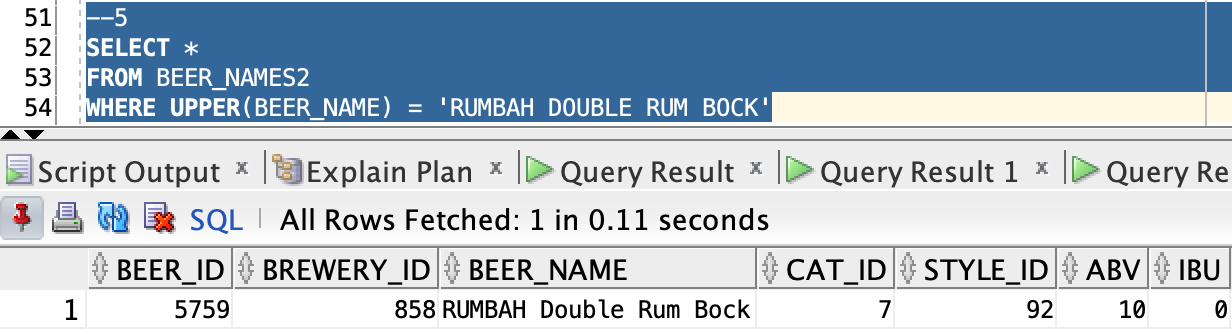


**Based on our experiments, we find that adding an index marginally improved results for point queries, reducing the time for retrieval by 0.009s in this case. However, adding an index did not improve performance for the range and scan queries.**

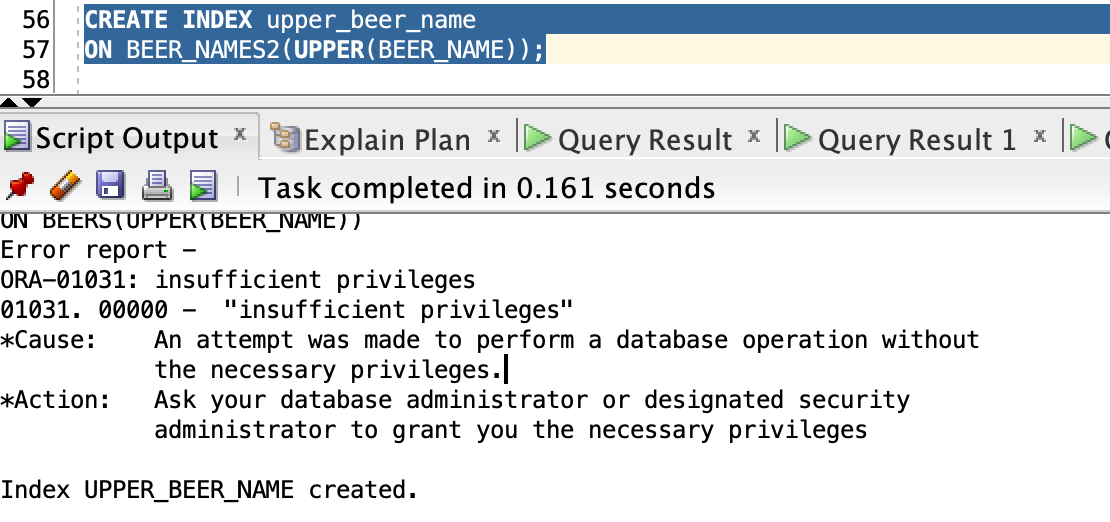
**This was an unexpected result which we investigated further and found that indexing often does not improve performance for range and scan queries as it forces the DB Engine to go through the index and then look up the data. In such cases a sequential read may be more optimized than an indexed read.**

## **Idea 5: Function-Based Indexes**

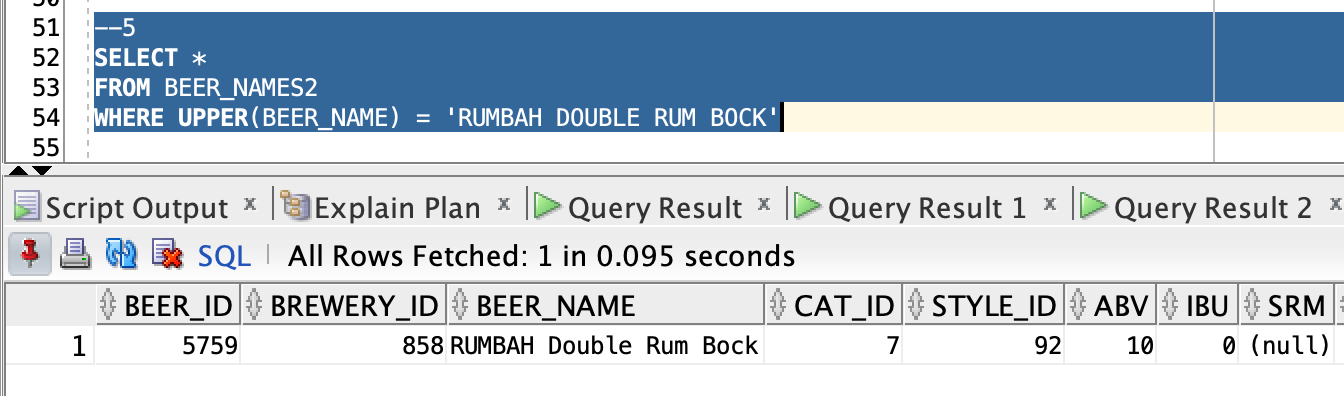
1. Before Function Based Indexing



1. Creating Function Based index for BEERS\_NAME



1. After Function Based Indexing



**Creating a simple function based index on BEER\_NAME produced a significant performance improvement from 0.161s to 0.095s a difference of ~40%**

**Idea 6: Database Programming**

You might also consider doing a bit of database programming based on the previous special topic.  For instance, you could write a stored procedure to generate data, such as fan ratings of movies (or craft beers).  You could also build utility procedures or functions for manipulating the data or computing derived information like a composite movie rating.

1. Stored Procedure for inserting records into the beer\_names1 table.

Graphical user interface, application

Description automatically generated

Here we are trying to insert a record into the beer\_names1 table. Running this stored procedure while providing values as in below image for the parameters in the parenthesis above inserts records into the beer\_names1 table as verified below.

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

1. Stored Procedure for retrieving the number of beers having a specific ABV value.

Graphical user interface, application, Word

Description automatically generated

Here we have a stored procedure which has one IN parameter and one OUT parameter. The IN parameter takes the value of the ABV whose number of beers we would like to know and the OUT NUM\_BEERS\_PARAM outputs the number of beers available with the given ABV.

Below screenshots show how to provide the input parameter value and how to verify the output.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

We can verify the answer by running a simple SELECT query.

Graphical user interface, text, application

Description automatically generated