**12 c New Features**

1. **Truncate table with cascase option**

Creating tables for Demo.

create table **my\_Category**(category\_id number,category\_name varchar2(50), constraint product\_id\_pk PRIMARY KEY(category\_id));

create table **my\_product**(product\_id number, product\_name varchar2(50), category\_id number,constraint product\_pk PRIMARY KEY(product\_id),constraint product\_Category\_fk foreign key(category\_id) references my\_Category(category\_id) on delete cascade);

insert into my\_Category values(1,'LED TV');

insert into my\_Category values(2,'LCD TV');

insert into my\_Category values(3,'SMART TV');

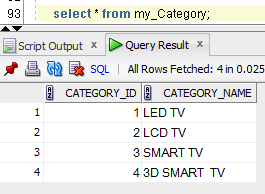
insert into my\_Category values(4,'3D SMART TV');

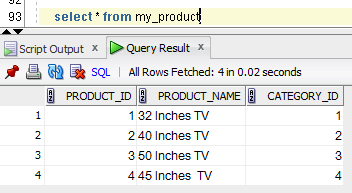
insert into my\_product values(1,'32 Inches TV',1);

insert into my\_product values(2,'40 Inches TV',2);

insert into my\_product values(3,'50 Inches TV',3);

insert into my\_product values(4,'45 Inches TV',4);

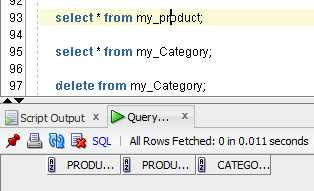




Now deleting from parent table

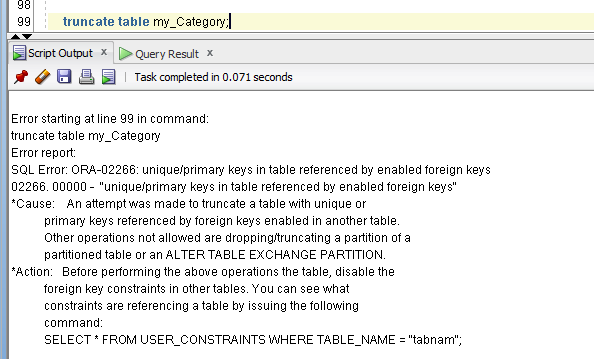
delete from my\_Category;

since table is created with on delete cascade data is deleted from child table as well.



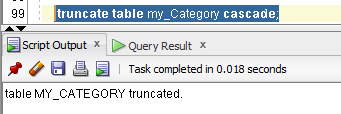
Now trying to truncate child table

Getting error as parent data is referred in child table.

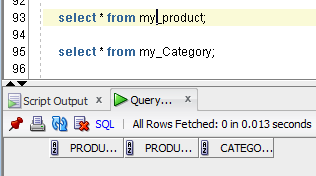


Now trying to truncate with cascade.

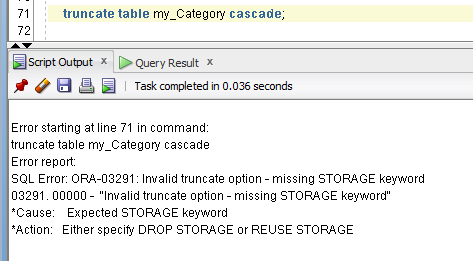
truncate table my\_Category cascade;



It worked same as delete with cascade.



Same command in oracle 11g throws errors



1. **Multiple indexes on same column**

Creating table for demo

create table my\_Category(category\_id number,category\_name varchar2(50));

insert into my\_Category values(1,'LED TV');

insert into my\_Category values(2,'LCD TV');

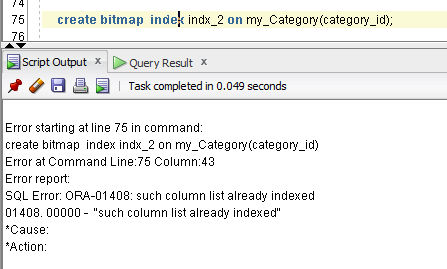
insert into my\_Category values(3,'SMART TV');

insert into my\_Category values(4,'3D SMART TV');

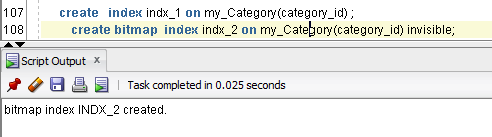
created normal index

create index indx\_1 on my\_Category(category\_id);

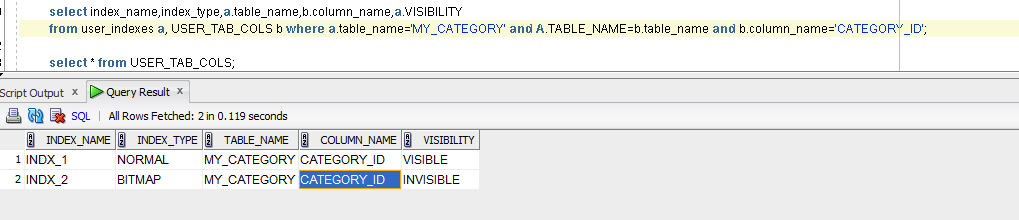
When we try to create another index on same column it throws error.



Samething we can do with invisible option

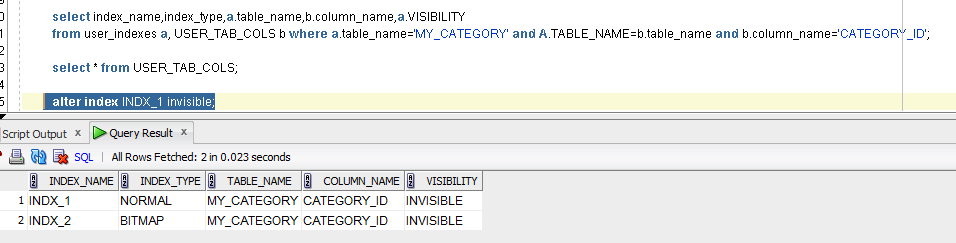


We can see from user\_indexes table same column has two indexes.



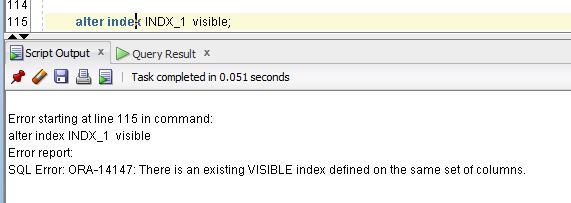
We can alter index to make it visible/invisible

alter index INDX\_1 invisible;



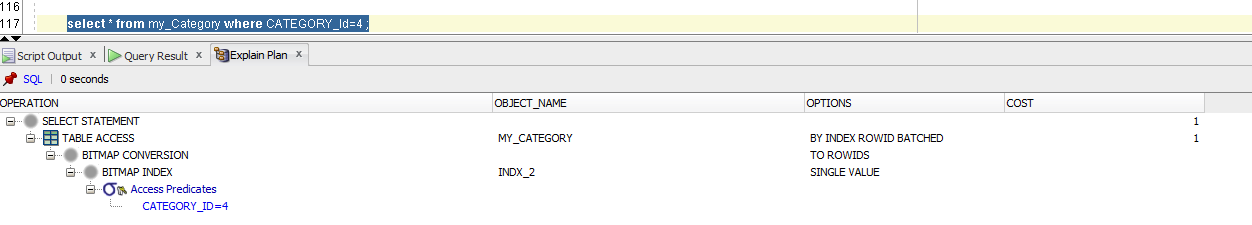
There can have only one visible index on column

If we try to make both visible will get below error

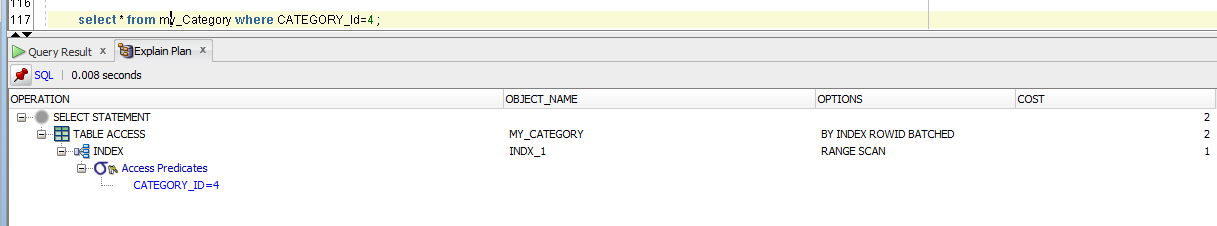


Now bitmap index is visible so optimizer s taking that index.

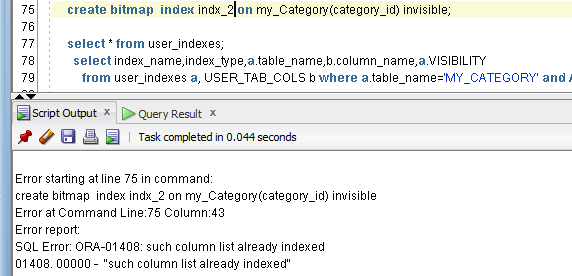
select \* from my\_Category where CATEGORY\_Id=4 ;



Now making normal index as visible , it is taking normal index



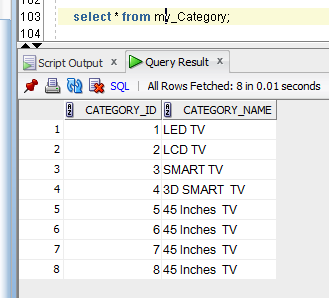
Samething if we try in 11g it throws error.



1. **APPROX\_COUNT\_DISTINCT**

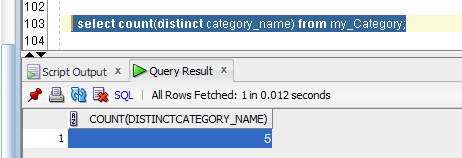
This function will return approximate distinct number of rows.

Here we can see records in table.

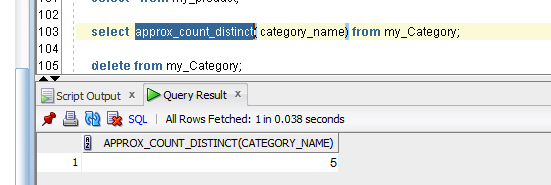


Now running below query to find the count of distinct record.

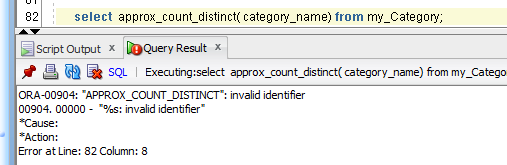
select count(distinct category\_name) from my\_Category;



Now replacing it with new command approx\_count\_distinct



**Same in 11g throws error**



1. **IDENTITY column for generating sequence value**

Normally we use sequence for generating unique values.

1n 12c we use identity column to generate new sequence values.

Below are options

1. Generated always as identity – used to generate unique values always.
2. Generated by default as identity
3. Generated by default on null as identity

Creating table for demo

create table my\_identity(id number generated always as identity,name varchar2(30));

insert into my\_identity(name) values ('AAA');

insert into my\_identity(name) values ('BBB');

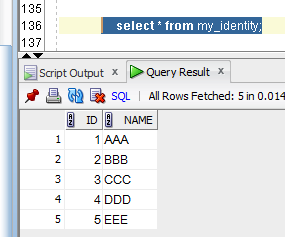
insert into my\_identity(name) values ('CCC');

insert into my\_identity(name) values ('DDD');

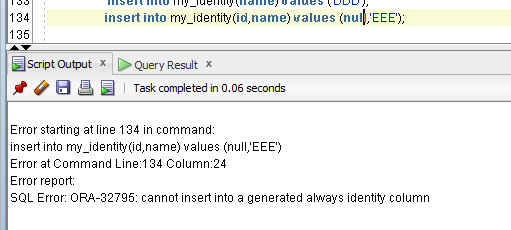
insert into my\_identity(name) values ('EEE');

now we will see data in table, unique values for id column will be generated.

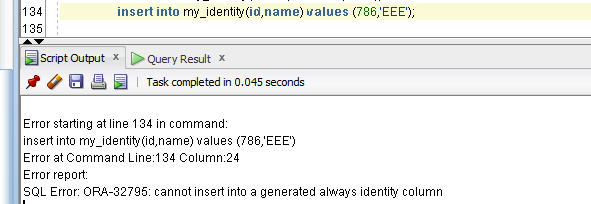
select \* from my\_identity;



It wont allow to insert null or default values with generated always as null

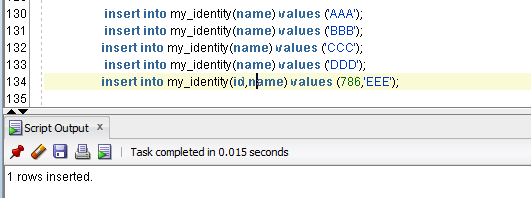


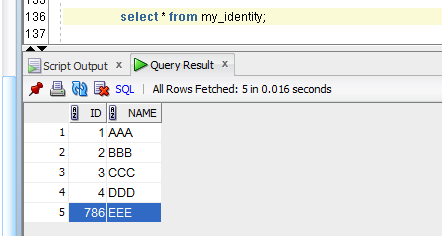
Trying with default values



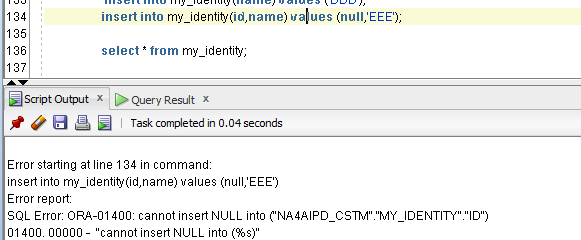
**Now we can achieve inserting default value by generated by default as identity**

create table my\_identity(id number **generated by default as identity**,name varchar2(30));



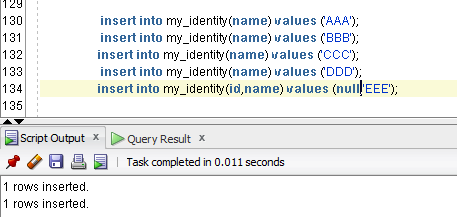


It wont allow to insert NULL values.

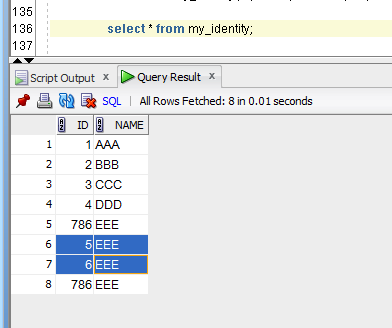


**Both can be overcome by using generated by default on null as identity**

create table my\_identity(id number **generated by default on null as identity**,name varchar2(30));

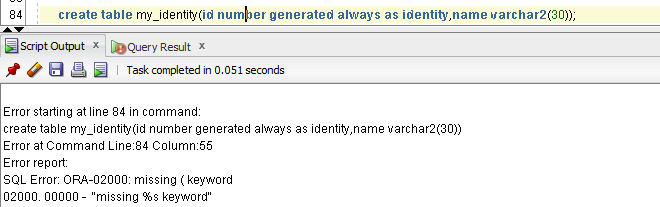


When we try to insert null unique values will be inserted.



Other than generated always as identity , other two will allow duplicates.

**Same in 11g gives error**



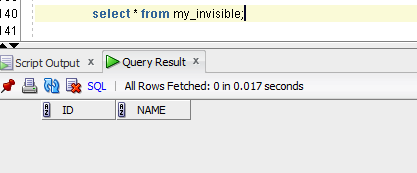
1. **Invisible columns**

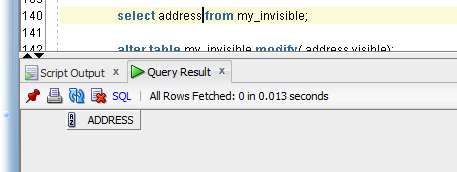
The column will be hidded . we can hide sensitive information.

Creating table for demo.

create table my\_invisible(id number,name varchar2(30),address varchar2(30) invisible);

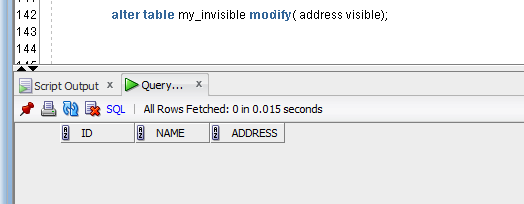
if we select that column from select \* from tablename, we cant see that column



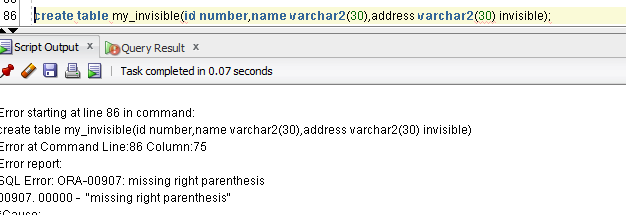


We can alter to make it visible

alter table my\_invisible modify( address visible);



**Same in 11g throws error**



1. **Creating partition on existing table.**

Till 11g below steps are taken to create partition on existing table.

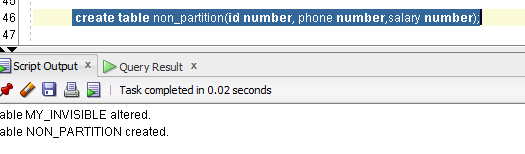
Creating new table with partition

Inserting data from old table to new table

Renaming original table to some archieve table

And renaming new table to original table.

create table non\_partition(id number, phone number,salary number);

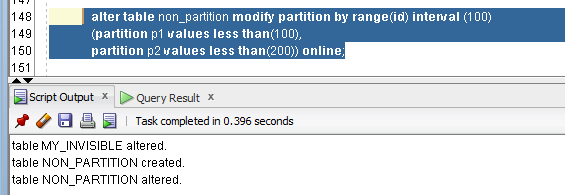


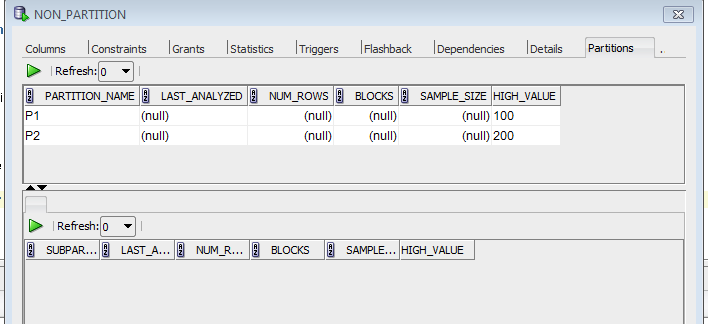
Adding partition

alter table non\_partition modify partition by range(id) interval (100)

(partition p1 values less than(100),

partition p2 values less than(200)) online;



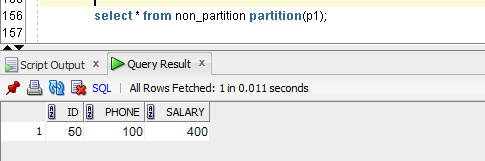


Now lets insert data to table

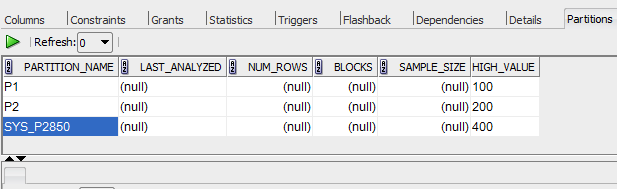
insert into non\_partition values(50,100,400);

insert into non\_partition values(190,100,400);

insert into non\_partition values(300,100,400);



Based on data new partition is created.



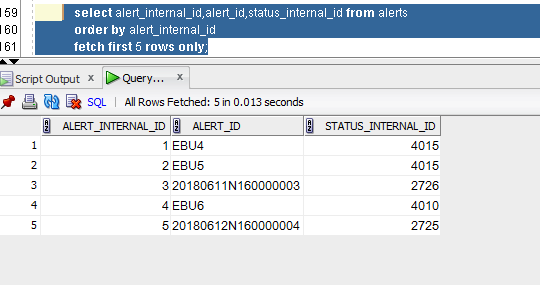
1. **ROW\_limiting\_clause**

**To fetch first 5 rows**

select alert\_internal\_id,alert\_id,status\_internal\_id from alerts

order by alert\_internal\_id

**fetch first 5 rows only;**



**To fetch 10 percent of data**

select alert\_internal\_id,alert\_id,status\_internal\_id from alerts

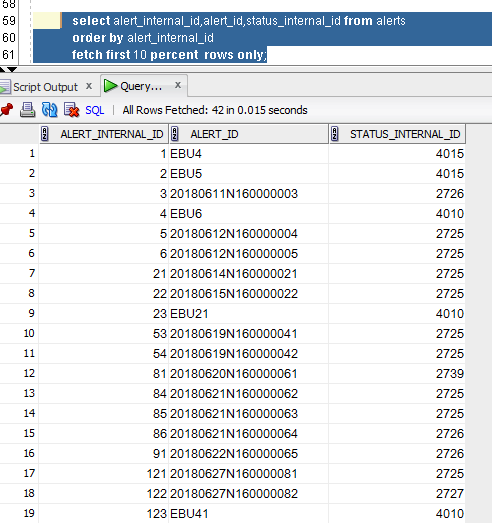
order by alert\_internal\_id

**fetch first 10 percent rows only;**

**or**

**select alert\_internal\_id,alert\_id,status\_internal\_id from alerts**

**fetch first 10 percent rows only;**

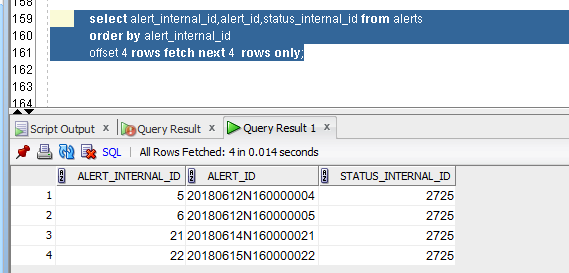


This query skip first 4 rows and disply next 4 rows(to fetch particily position)

select alert\_internal\_id,alert\_id,status\_internal\_id from alerts

order by alert\_internal\_id

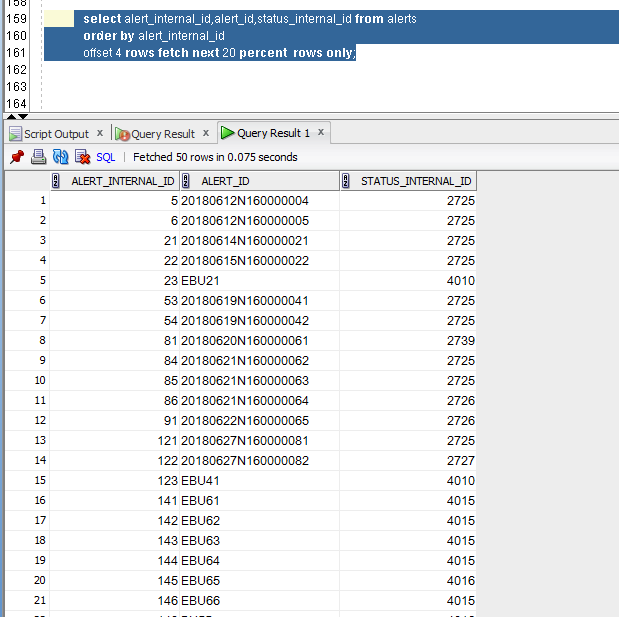
**offset 4 rows fetch next 4 rows only;**



**select alert\_internal\_id,alert\_id,status\_internal\_id from alerts**

**order by alert\_internal\_id**

**offset 4 rows fetch next 20 percent rows only;**

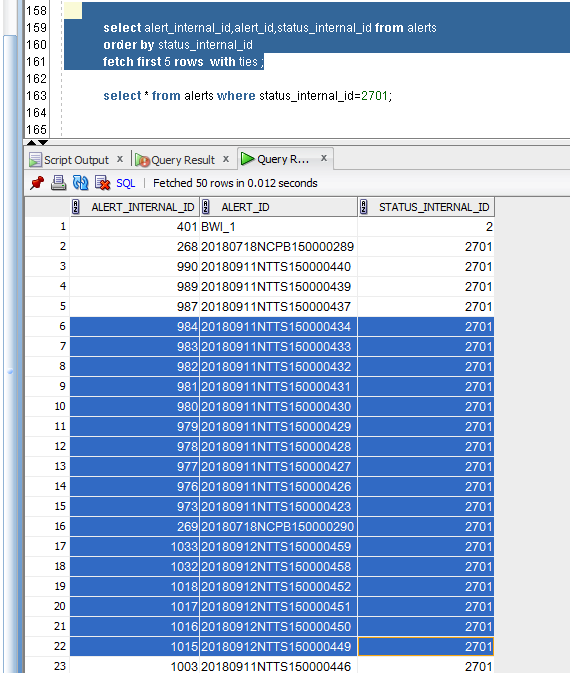


**select alert\_internal\_id,alert\_id,status\_internal\_id from alerts**

**order by status\_internal\_id**

**fetch first 5 rows with ties ;**

when using ties order by is compulsory

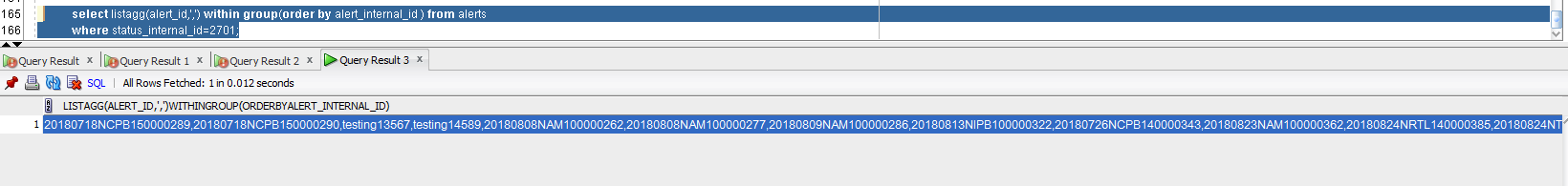


After 5th row is data is there with same value all records will be displayed.

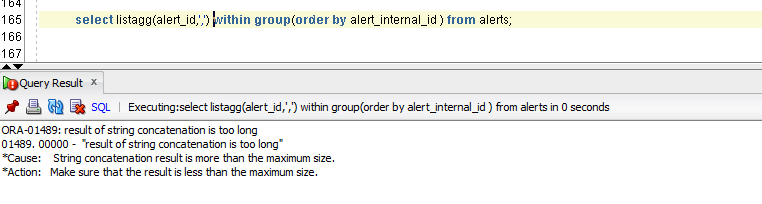
1. **Listagg function in 12c**

select listagg(alert\_id,',') within group(order by alert\_internal\_id ) from alerts

where status\_internal\_id=2701;

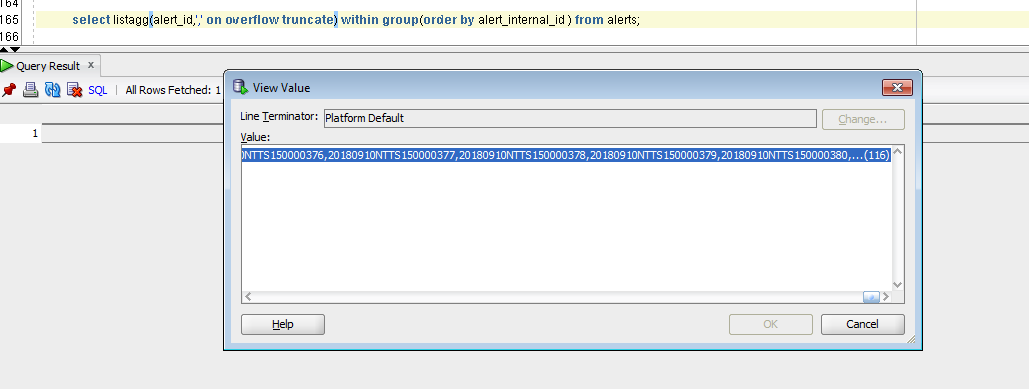


Listagg will throw error if size is more than 4000 bytes



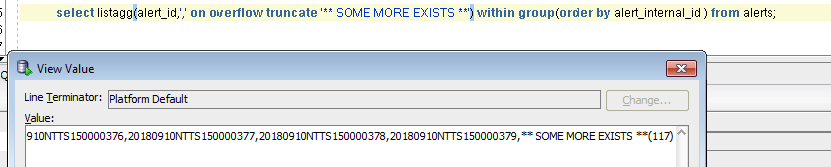
We can over by below option

select listagg(alert\_id,',' **on overflow truncate**) within group(order by alert\_internal\_id ) from alerts;



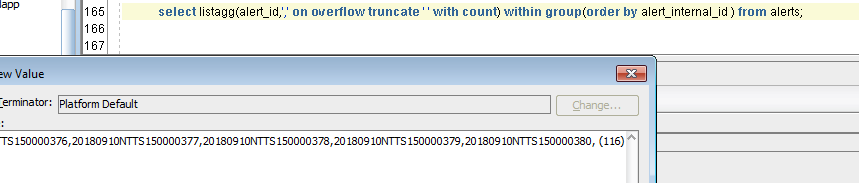
We can give more option like this

select listagg(alert\_id,',' on overflow truncate '\*\* SOME MORE EXISTS \*\*') within group(order by alert\_internal\_id ) from alerts;



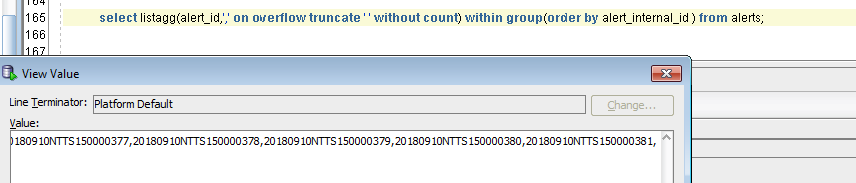
Just to see count with space

select listagg(alert\_id,',' on overflow truncate ' ' with count) within group(order by alert\_internal\_id ) from alerts;

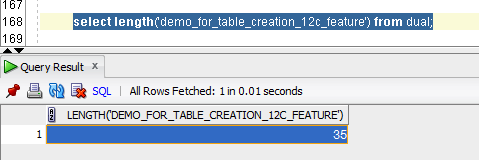


Just to see without count

select listagg(alert\_id,',' on overflow truncate ' ' without count) within group(order by alert\_internal\_id ) from alerts;

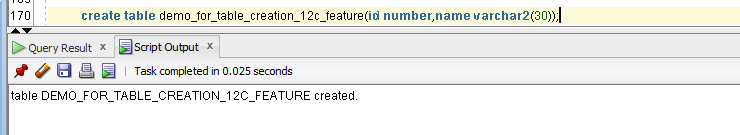


1. **Creating table/sequence names with more than 30 characters**



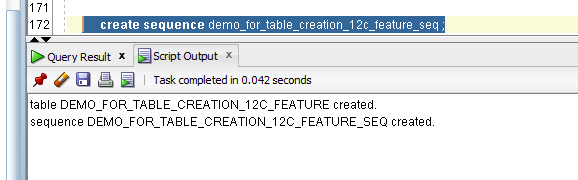
Creating table with that name

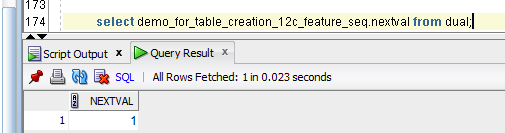
create table demo\_for\_table\_creation\_12c\_feature(id number,name varchar2(30));



Now creating sequence

create sequence demo\_for\_table\_creation\_12c\_feature\_seq ;





Even view we can create

create view demo\_for\_table\_creation\_12c\_feature\_view as select \* from demo\_for\_table\_creation\_12c\_feature;

view DEMO\_FOR\_TABLE\_CREATION\_12C\_FEATURE\_VIEW created.

1. **With clause with functions**

WITH

FUNCTION findLatestEnrolmentDate IS

.BEGIN

SELECT MAX(enrolment\_date)

FROM student;

END;

SELECT first\_name, last\_name

FROM student

WHERE enrolment\_date = findLatestEnrolmentDate;

1. **DDL LOGGING**

Alter system set enable\_ddl\_logging=TRUE;

Alter session set enable\_ddl\_logging=TRUE;

We can see log whenever ddl command runs under path oracle\_home\_directory

Eg

\diag\rdbms\orcl12c\orcl12c\log\ddl

1. **PRAGMA UDF**

**Pragma udf tells the compiler that the plsql unit is a user defined function that is used primarily in SQL statements , which might improves its performance**

**create or replace function fn1(input\_num number) return number as**

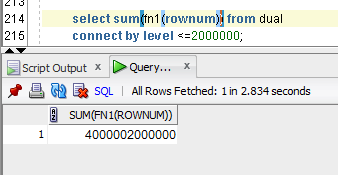
**pragma udf;**

**begin**

**return input\_num\*2;**

**end ;**

**we can see it took approx. 2.8 sec**



Created function without pragma

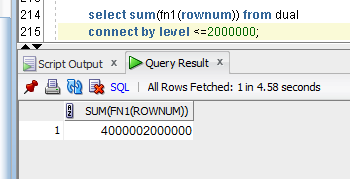
create or replace function fn1(input\_num number) return number as

begin

return input\_num\*2;

end ;

/



**It took around 4.57 seconds**

**Pragma udf increases the performance**