Oracle SQL Tuning Basics Tips and Tricks

Description:

BISP is committed to provide BEST learning material to the beginners and advance learners. In the same series, we have prepared a complete end-to end Hands-on Guide SQL optimization tips. The document focuses on top 21 tics for SQL optimization. See our youtube collections for more details. Join our professional training program and learn from experts.

History:

Version	Description Change	Author	Publish Date
0.1	Initial Draft	Kuldeep Mishra	12th Aug 2011
0.1	Review#1	Amit Sharma	18th Aug 2011

Contents

Contents....2
Tuning Tips#1....3
Tuning Tips#2....4
Tuning Tips#3....4
Tuning Tips#4....5
Tuning Tips#5....6
Tuning Tips#6....6
Tuning Tips#7....8
Tuning Tips#8....11
Tuning Tips#9....12
Tuning Tips#10....13
Tuning Tips#11....15
Tuning Tips#12....16
Tuning Tips#13....17
Tuning Tips#14....17
Tuning Tips#15....18
Tuning Tips#16....18
Tuning Tips#17....19
Tuning Tips#18....19
Tuning Tips#19....20
Tuning Tips#20....21

***Objective*:** SQL Tuning top 20 Rules for Oracle SQL beginners and intermediate learners.

Oracle SQL Tuning Tips

Consideration when writing an SQL statement is that it returns a correct result. The second is that it be the most efficient for a given situation. You can use many different SQL statements to achieve the same result. It is often the case that only one statement will be the most efficient choice in a given situation.

Remember that processing SQL is a sequence of Parse (syntax check and object resolution), Execution (required reads and writes), and Fetch (row results retrieved, listed, sorted, and returned). SQL "tuning" consists, quite simply, of reducing one or more of them.

Note: generally Parse is the greatest time and resource hog. Parse overhead can be minimized by the use of Procedures, Functions, Packages, Views, etc.

Inadequate performance can have a significant cost impact on your business. A poor performing system and application can result in customer dissatisfaction, reduced productivity, and high costs. It is absolutely critical that the system's performance is operating at its peak levels.

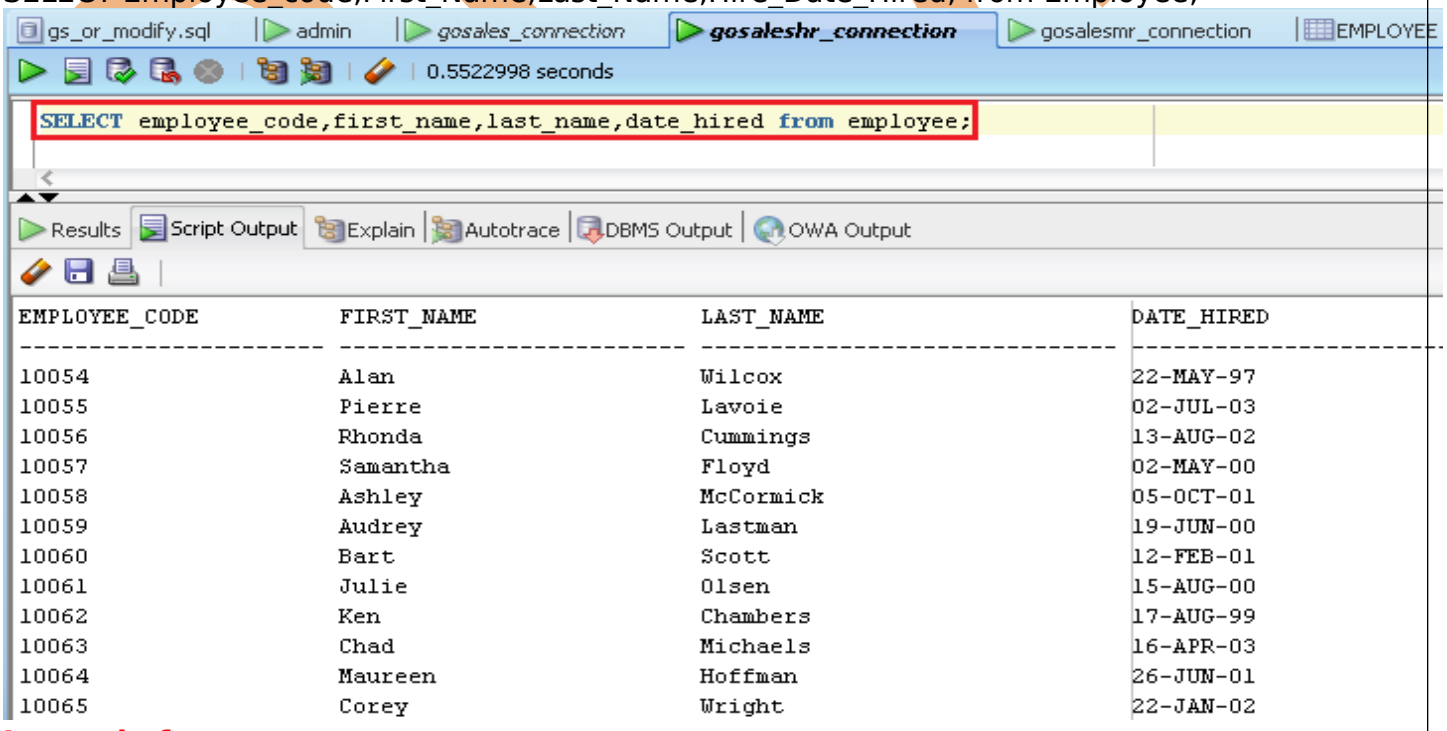
Following are some general tips that often increase SQL statement efficiency. Being general they may not apply to a particular scenario.

SQL Tuning Top 20 Tics

Tuning Tips#1

1) The sql query becomes faster if you use the actual columns names in SELECT statement instead of than '*'.

SELECT Employee_code,First_Name,Last_Name,Hire_Date_Hired, from Employee;



EMPLOYEE_CODE	FIRST_NAME	LAST_NAME	DATE_HIRED
10054	Alan	Wilcox	22-MAY-97
10055	Pierre	Lavoie	02-JUL-03
10056	Rhonda	Cummings	13-AUG-02
10057	Samantha	Floyd	02-MAY-00
10058	Ashley	McCormick	05-OCT-01
10059	Audrey	Lastman	19-JUN-00
10060	Bart	Scott	12-FEB-01
10061	Julie	Olsen	15-AUG-00
10062	Ken	Chambers	17-AUG-99
10063	Chad	Michaels	16-APR-03
10064	Maureen	Hoffman	26-JUN-01
10065	Corey	Wright	22-JAN-02

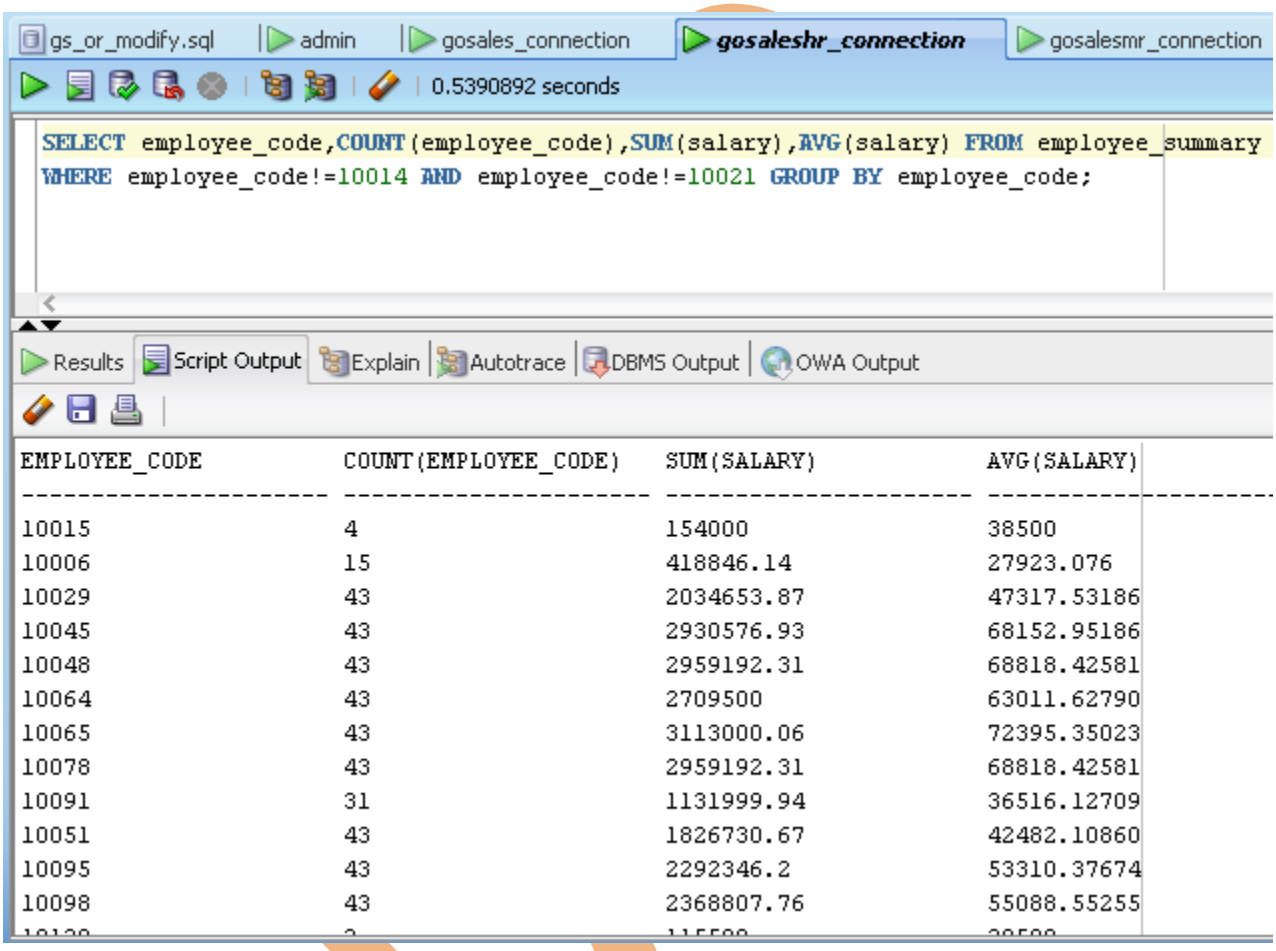
Instead of :

SELECT * from Employees;

Tuning Tips#2

2) HAVING clause is used to filter the rows after all the rows are selected. It is just like a filter. Do not use HAVING clause for any other purposes.
For Example: Write the query as

SELECT employee_code,COUNT(employee_code),SUM(salary),AVG(salary) FROM employee_summary WHERE employee_code!=10014 AND employee_code!=10021 GROUP BY employee_code;

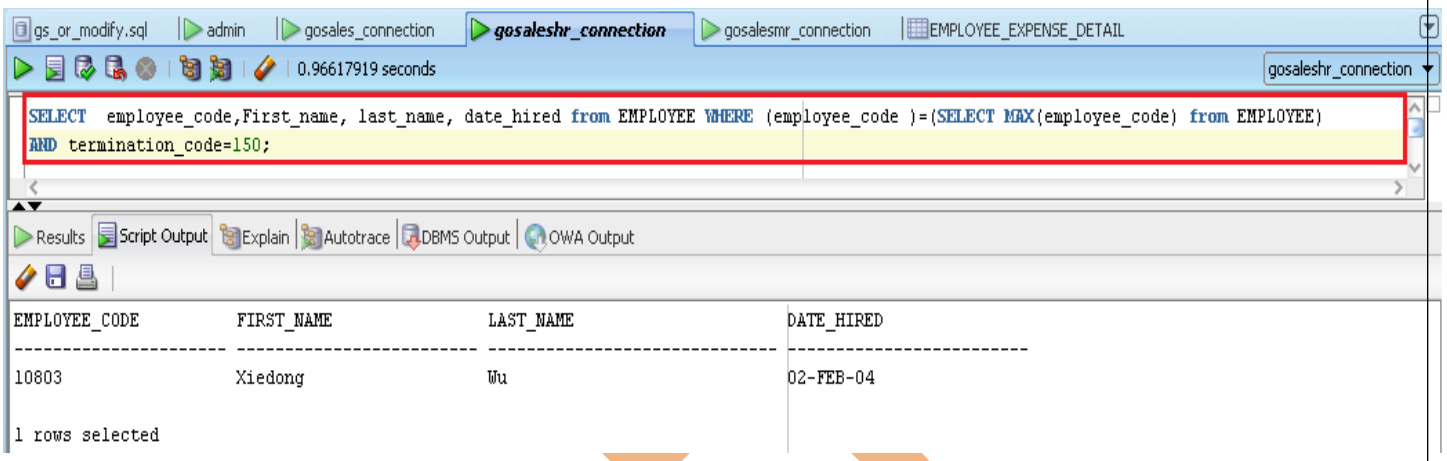


Instead of:

SELECT employee_code,COUNT(employee_code),SUM(salary),AVG(salary) FROM employee_summary GROUP BY employee_code Having employee_code!=10014 AND employee_code!=10021 ;

Tuning Tips#3

Sometimes you may have more than one subqueries in your main query. Try to minimize the number of subquery block in your query.
For Example: Write the query as
SELECT employee_code,First_name, last_name, date_hired from EMPLOYEE WHERE (employee_code)=(SELECT MAX(employee_code) from EMPLOYEE)
AND termination_code=150;



Instead of:

SELECT employee_code,first_name, last_name, date_hired from EMPLOYEE WHERE employee_code =(SELECT MAX(employee_code) from EMPLOYEE) AND termination_code=150;

Tuning Tips#4

Use operator EXISTS, IN and table joins appropriately in your query.

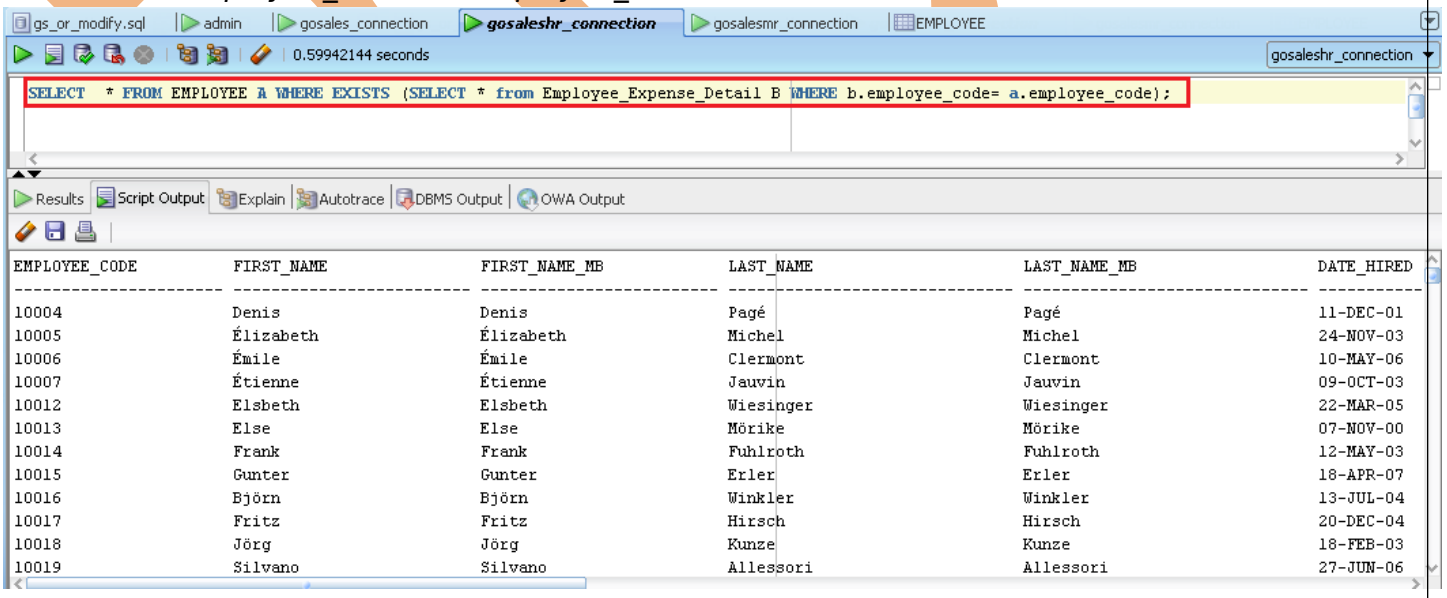
a) Usually IN has the slowest performance.

b) IN is efficient when most of the filter criteria is in the sub-query.

c) EXISTS is efficient when most of the filter criteria is in the main query.

For Example: Write the query as

*SELECT * from EMPLOYEE A WHERE EXISTS (SELECT * from Employee_Expense_Detail B WHERE b.employee_code= a.employee_code);*



Instead of:

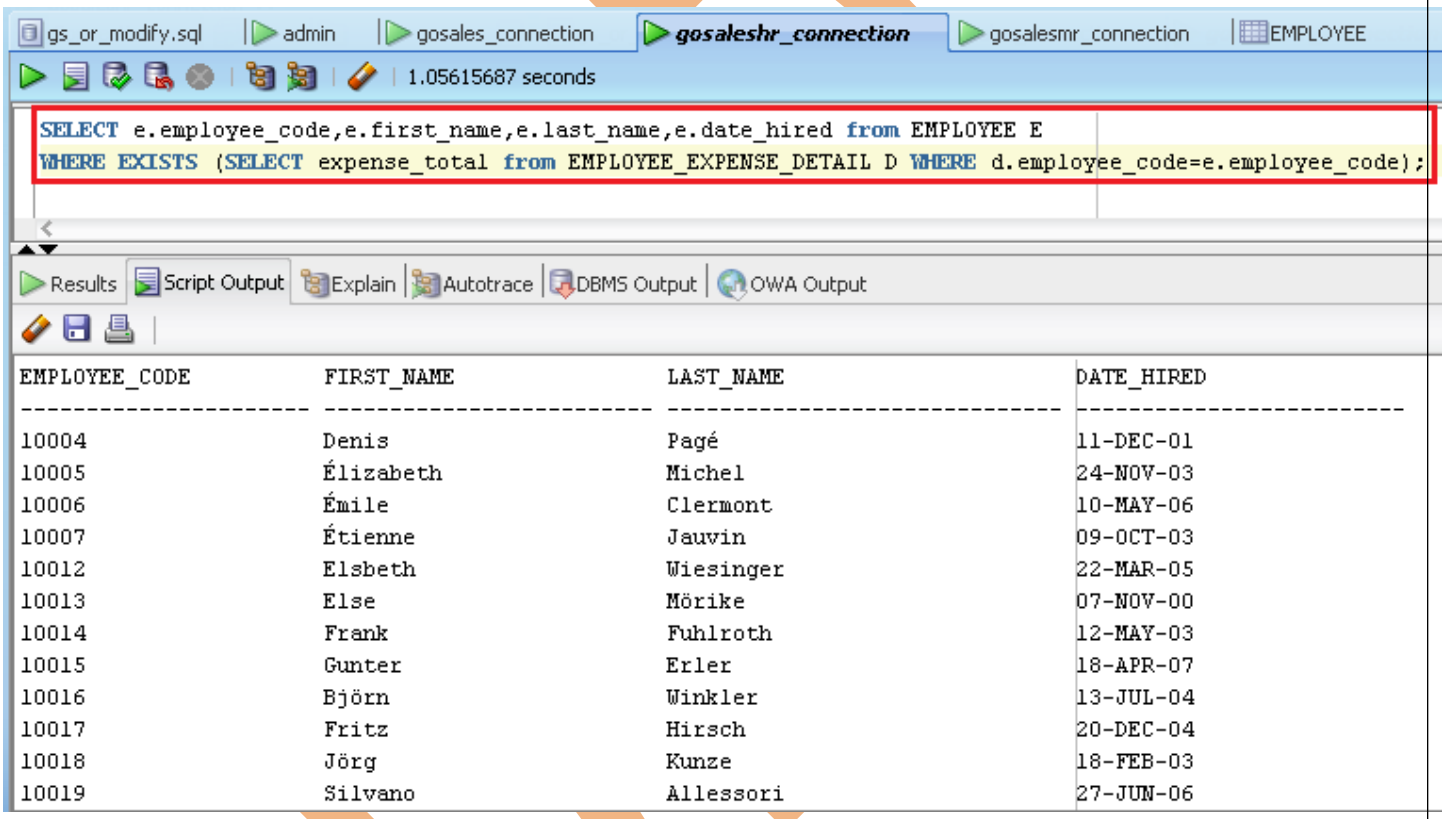
*SELECT * from EMPLOYEE A WHERE employee_code IN(SELECT employee_code from Employee_Expense_Detail);*

Tuning Tips#5

Use EXISTS instead of DISTINCT when using joins which involves tables having one-to-many relationship.

For Example: Write the query as

SELECT e.employee_code,e.first_name,e.last_name,e.date_hired from EMPLOYEE E WHERE EXISTS (SELECT expense_total from EMPLOYEE_EXPENSE_DETAIL D WHERE d.employee_code=e.employee_code);



EMPLOYEE_CODE	FIRST_NAME	LAST_NAME	DATE_HIRED
10004	Denis	Pagé	11-DEC-01
10005	Élizabeth	Michel	24-NOV-03
10006	Émile	Clermont	10-MAY-06
10007	Étienne	Jauvin	09-OCT-03
10012	Elsbeth	Wiesinger	22-MAR-05
10013	Else	Mörike	07-NOV-00
10014	Frank	Fuhlroth	12-MAY-03
10015	Gunter	Erler	18-APR-07
10016	Björn	Winkler	13-JUL-04
10017	Fritz	Hirsch	20-DEC-04
10018	Jörg	Kunze	18-FEB-03
10019	Silvano	Allessori	27-JUN-06

Instead of:

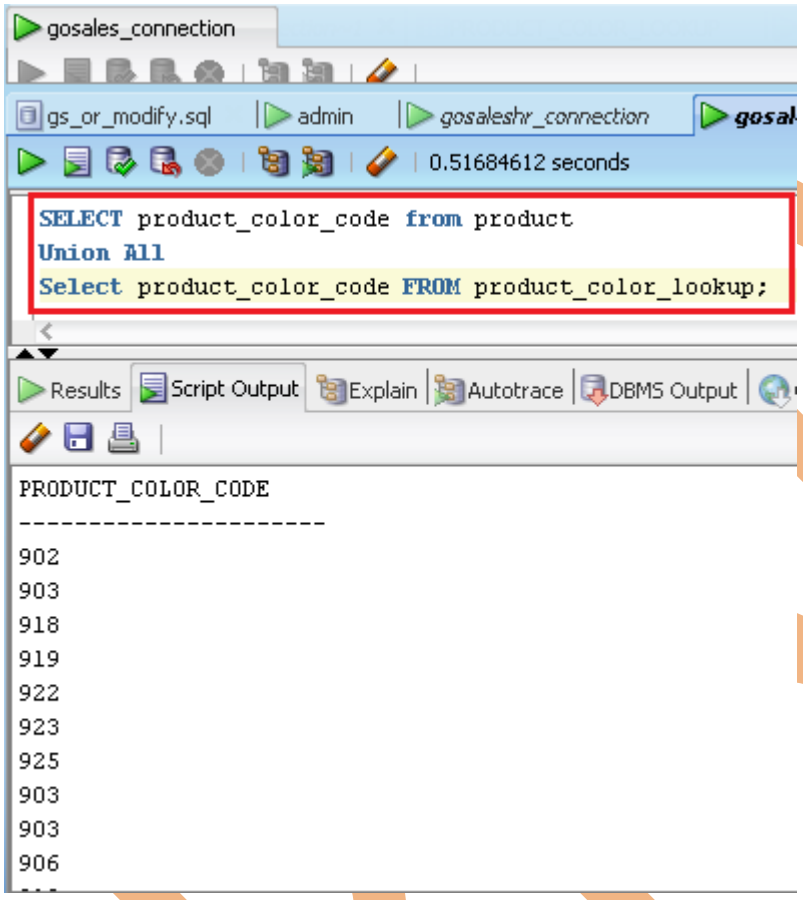
SELECT DISTINCT e.employee_code,e.first_name,e.last_name,e.date_hired from EMPLOYEE E,EMPLOYEE_EXPENSE_DETAIL D WHERE d.employee_code=e.employee_code;

Tuning Tips#6

Try to use UNION ALL in place of UNION.

For Example: Write the query as

SELECT product_color_code from product Union All Select product_color_code from product_color_lookup;

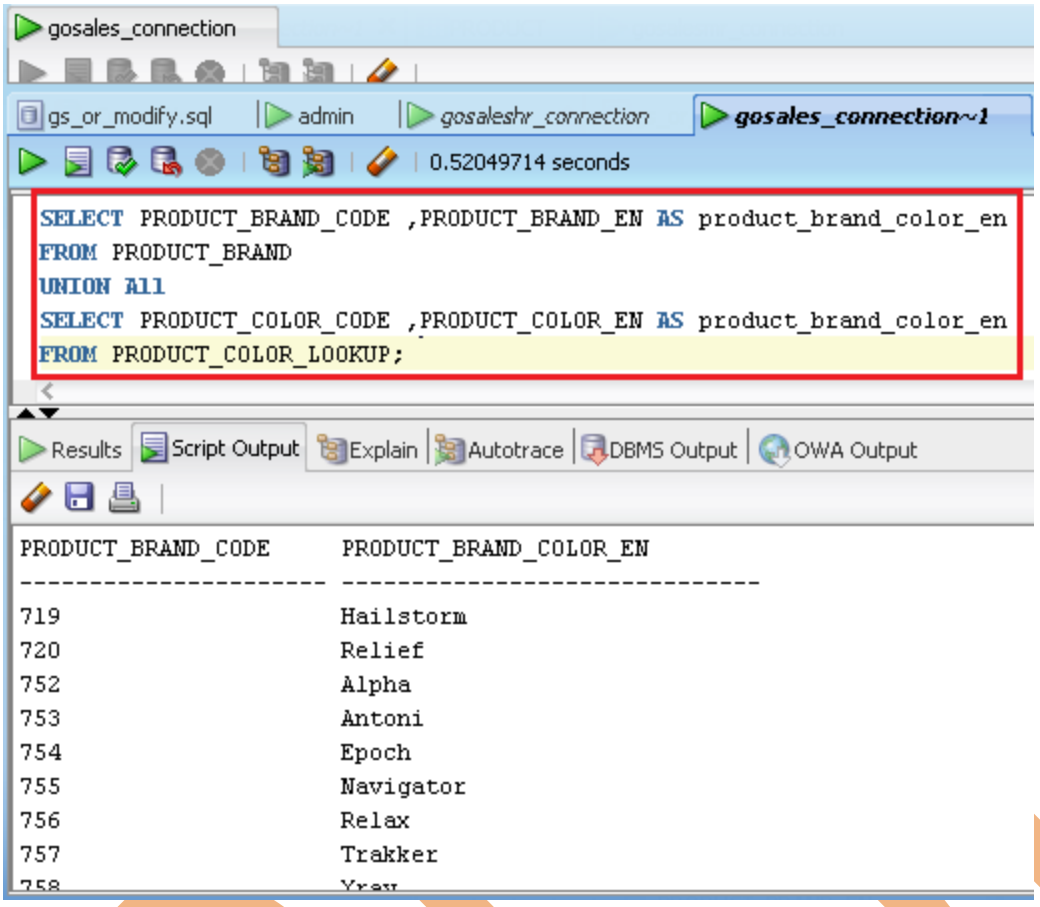


Instead of:

SELECT product_color_code from product Union Select product_color_code from product_color_lookup;

For Example: Write the query as

SELECT product_brand_code ,product_brand_en AS product_brand_color_en FROM PRODUCT_BRAND UNION All SELECT product_color_code ,product_color_en AS product_brand_color_en from PRODUCT_COLOR_LOOKUP;



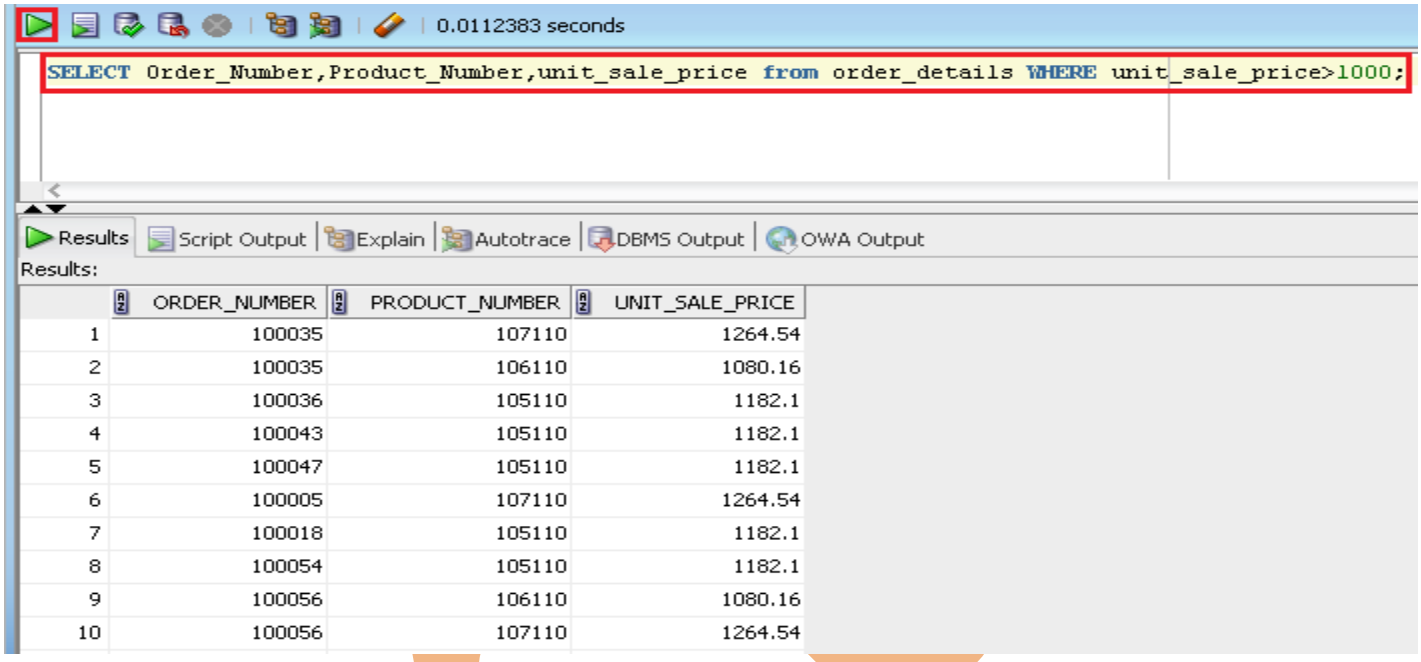
Instead of:

SELECT product_brand_code ,product_brand_en AS product_brand_color_en FROM PRODUCT_BRAND UNION SELECT product_color_code ,product_color_en AS product_brand_color_en from PRODUCT_COLOR_LOOKUP;

Tuning Tips#7

Beware of WHERE clauses which do not use indexes at all. Even if there is an index over a column that is referenced by a WHERE clause included in this section, Oracle will ignore the index. All these WHERE clause can be re-written to use an index while returning the same values. In other words, Do not perform operations on database objects referenced in the WHERE clause:

SELECT order_number,product_number,unit_sale_price from order_details WHERE unit_sale_price>1000;



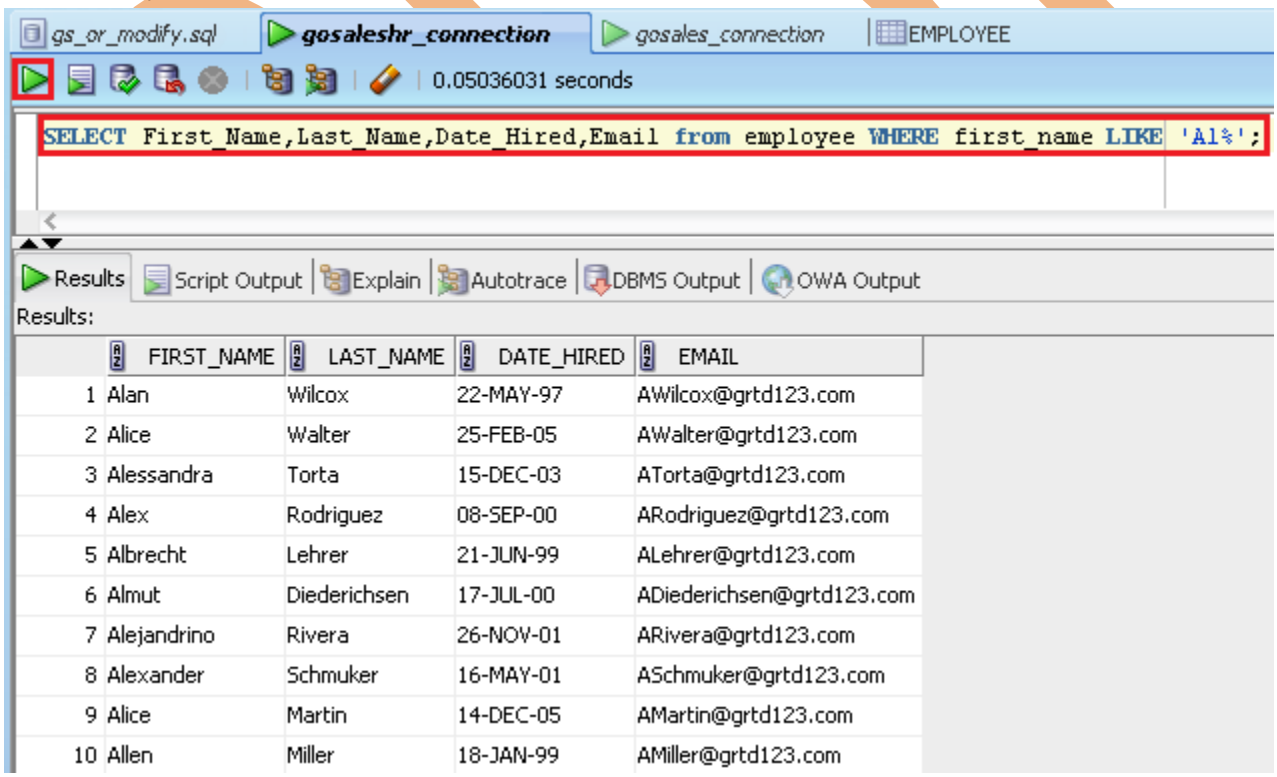
	ORDER_NUMBER	PRODUCT_NUMBER	UNIT_SALE_PRICE
1	100035	107110	1264.54
2	100035	106110	1080.16
3	100036	105110	1182.1
4	100043	105110	1182.1
5	100047	105110	1182.1
6	100005	107110	1264.54
7	100018	105110	1182.1
8	100054	105110	1182.1
9	100056	106110	1080.16
10	100056	107110	1264.54

Rather Than,

SELECT order_number,product_number,unit_sale_price from order_details WHERE unit_sale_price!=1000;

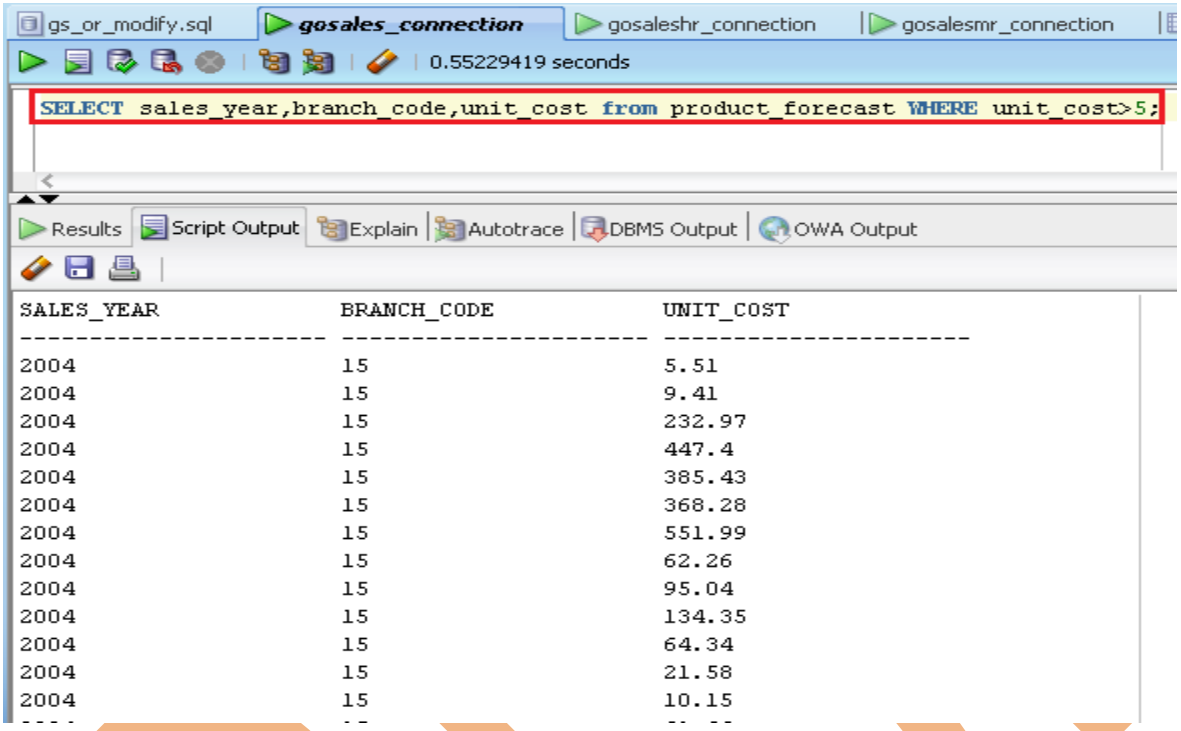
USE:

SELECT First_Name,Last_Name,Date_Hired,Email from employee WHERE first_name LIKE 'Al%';



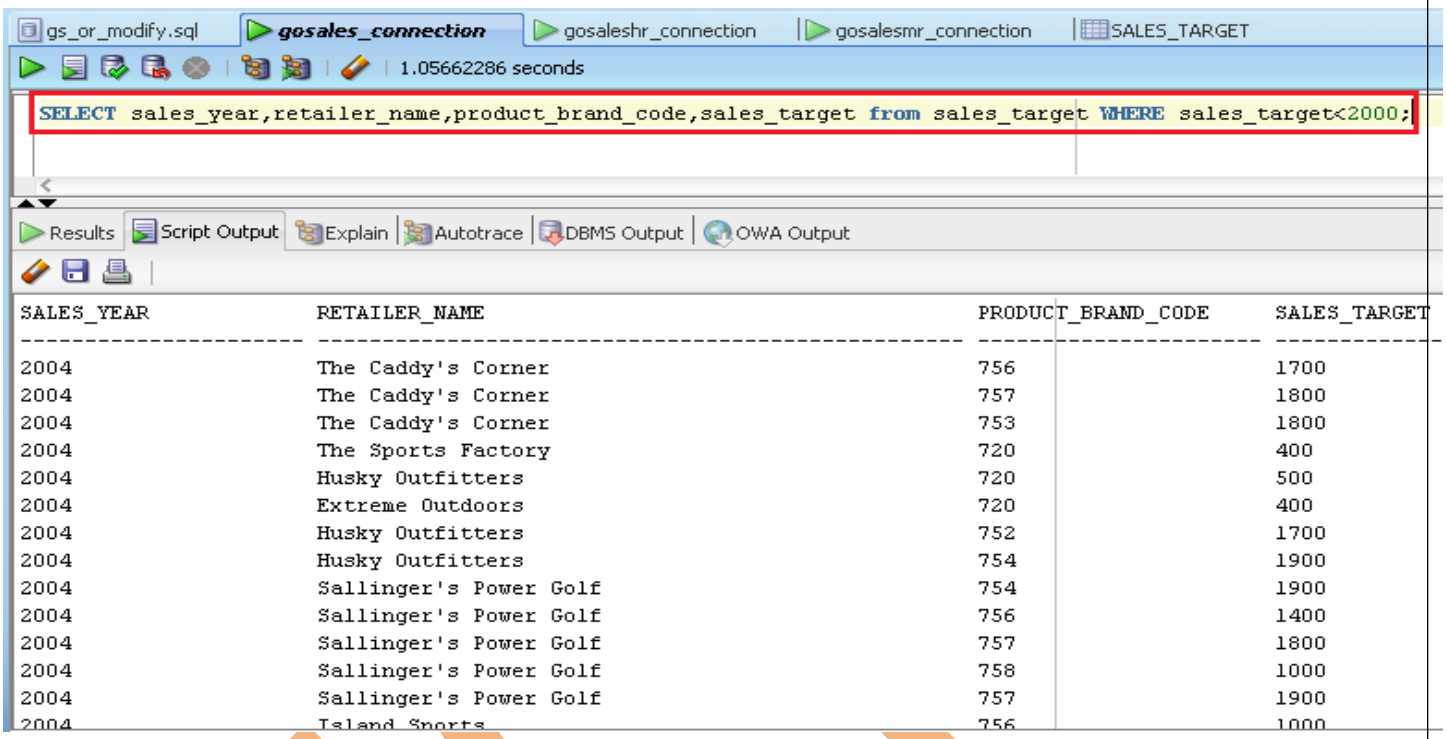
	FIRST_NAME	LAST_NAME	DATE_HIRED	EMAIL
1	Alan	Wilcox	22-MAY-97	AWilcox@grtd123.com
2	Alice	Walter	25-FEB-05	AWalter@grtd123.com
3	Alessandra	Torta	15-DEC-03	ATorta@grtd123.com
4	Alex	Rodriguez	08-SEP-00	ARodriguez@grtd123.com
5	Albrecht	Lehrer	21-JUN-99	ALehrer@grtd123.com
6	Almut	Diederichsen	17-JUL-00	ADiederichsen@grtd123.com
7	Alejandrino	Rivera	26-NOV-01	ARivera@grtd123.com
8	Alexander	Schmuker	16-MAY-01	ASchmuker@grtd123.com
9	Alice	Martin	14-DEC-05	AMartin@grtd123.com
10	Allen	Miller	18-JAN-99	AMiller@grtd123.com

Rather Than,
SELECT First_Name,Last_Name,Date_Hired,Email from employee WHERE SUBSTR(first_name,1,2)='Al';
USE:
SELECT sales_year,branch_code,unit_cost from product_forecast WHERE unit_cost >5;



SALES_YEAR	BRANCH_CODE	UNIT_COST
2004	15	5.51
2004	15	9.41
2004	15	232.97
2004	15	447.4
2004	15	385.43
2004	15	368.28
2004	15	551.99
2004	15	62.26
2004	15	95.04
2004	15	134.35
2004	15	64.34
2004	15	21.58
2004	15	10.15

Instead of :
SELECT sales_year,branch_code,unit_cost from product_forecast WHERE unit_cost NOT= 5;
USE:
SELECT sales_year,retailer_name,product_brand_code,sales_target from sales_target WHERE sales_target<2000;

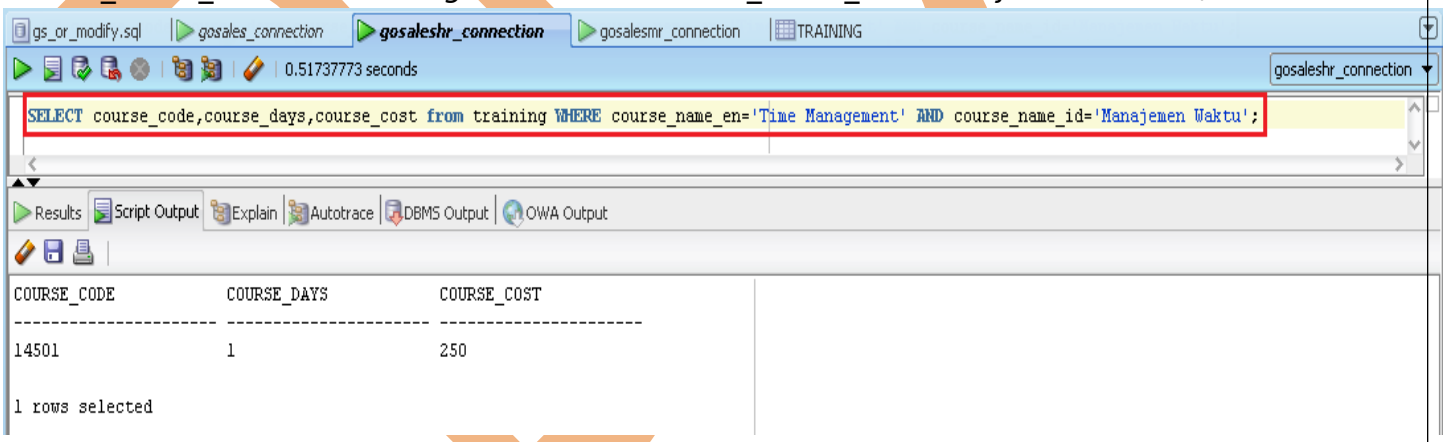


Instead of :

SELECT sales_year,retailer_name,product_brand_code,sales_target from sales_target WHERE sales_target + 3000<5000;

USE:

SELECT course_code,course_days,course_cost from training WHERE course_name_en='Time Management' AND course_name_id='Manajemen Waktu';



Instead of :

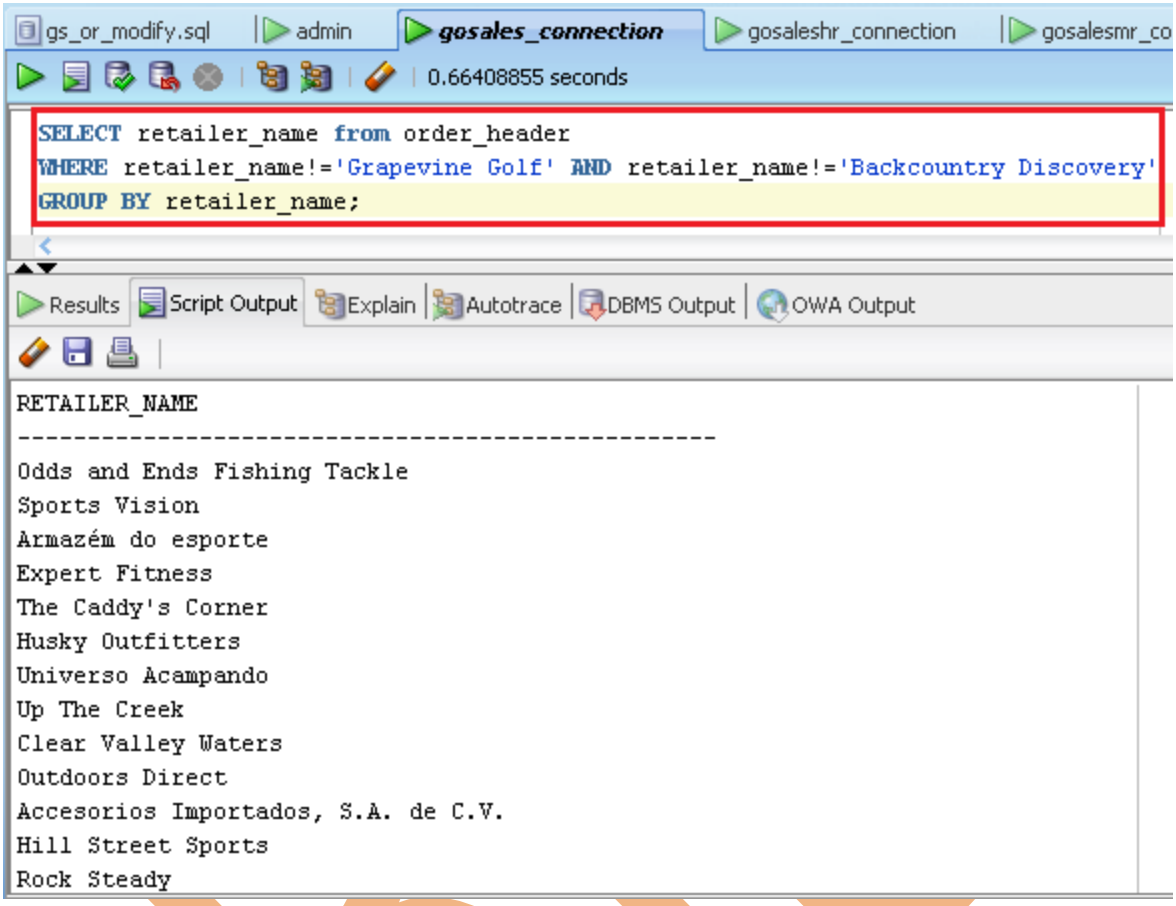
SELECT course_code,course_days,course_cost from training WHERE course_name_en || course_name_id= 'Time Management Manajemen Waktu';

Tuning Tips#8

Don't forget to tune views. Views are SELECT statements and can be tuned in just the same way as any other type of SLECT statement can be. All tuning applicable to any SQL statement are equally applicable to views.

Avoid including a HAVING clause in SELECT statements. The HAVING clause filters selected rows only after all rows have been fetched. Using a WHERE clause helps

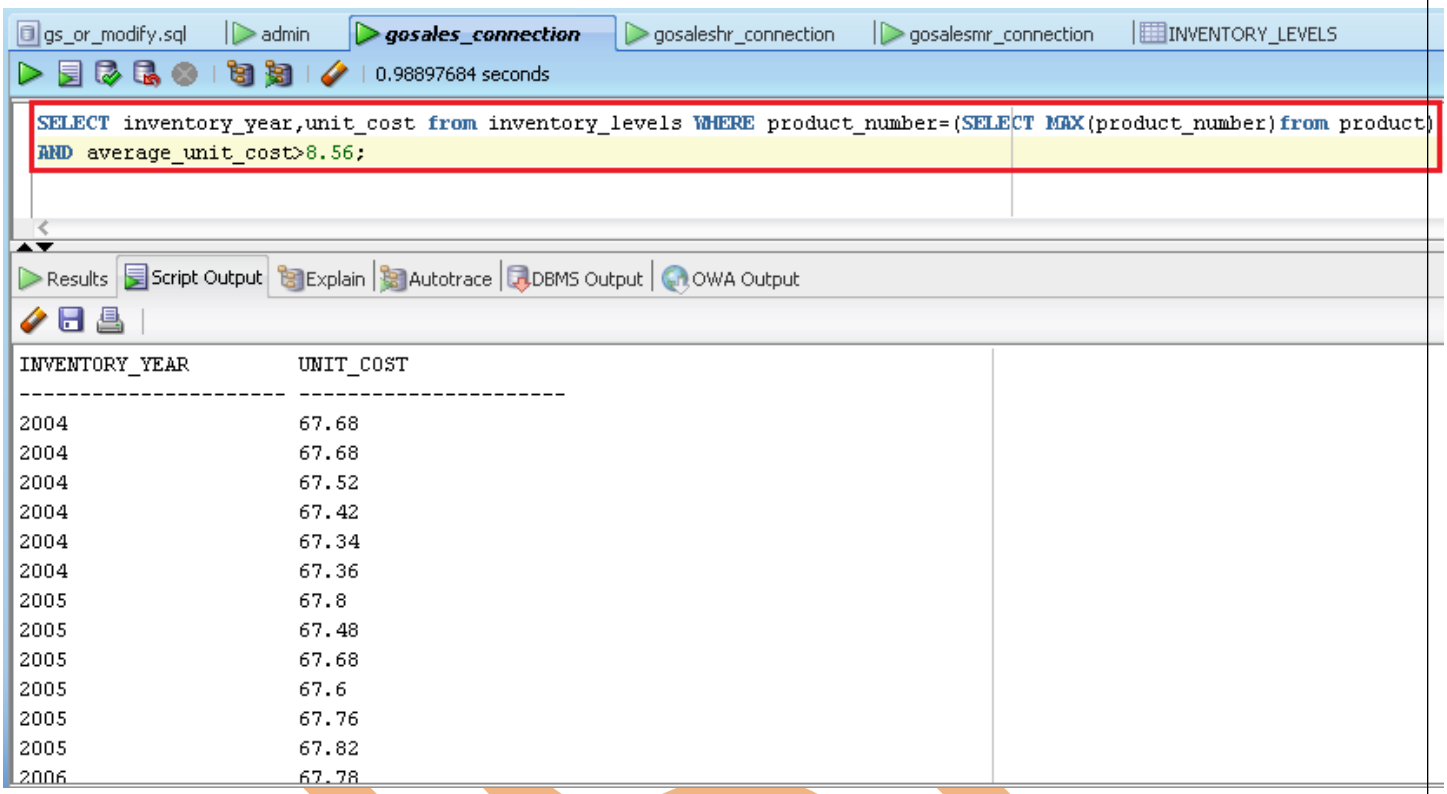
reduce overheads in sorting, summing, etc. HAVING clauses should only be used when columns with summary operations applied to them are restricted by the clause.
USE:
SELECT retailer_name from order_header WHERE retailer_name!='Grapevine Golf' AND retailer_name!='Backcountry Discovery' GROUP BY retailer_name;



Instead of :
SELECT retailer_name from order_header GROUP BY retailer_name HAVING retailer_name!='Grapevine Golf' AND retailer_name!='Backcountry Discovery';

Tuning Tips#9

Minimize the number of table lookups (subquery blocks) in queries, particularly if our statements include subquery SELECTs or multicolumn UPDATEs.
USE:
SELECT inventory_year,unit_cost from inventory_levels WHERE product_number=(SELECT MAX(product_number) from product) AND average_unit_cost>8.5



Tuning Tips#10

When writing a sub-query (a SELECT statement within the WHERE or HAVING clause of another SQL statement):

i) Use a correlated (refers to at least one value from the outer query) sub-query when the return is relatively small and/or other criteria are efficient i.e. if the tables within the sub-query have efficient indexes.

ii) Use a non-correlated (does not refer to the outer query) sub-query when dealing with large tables from which you expect a large return (many rows) and/or if the tables within the sub-query do not have efficient indexes.

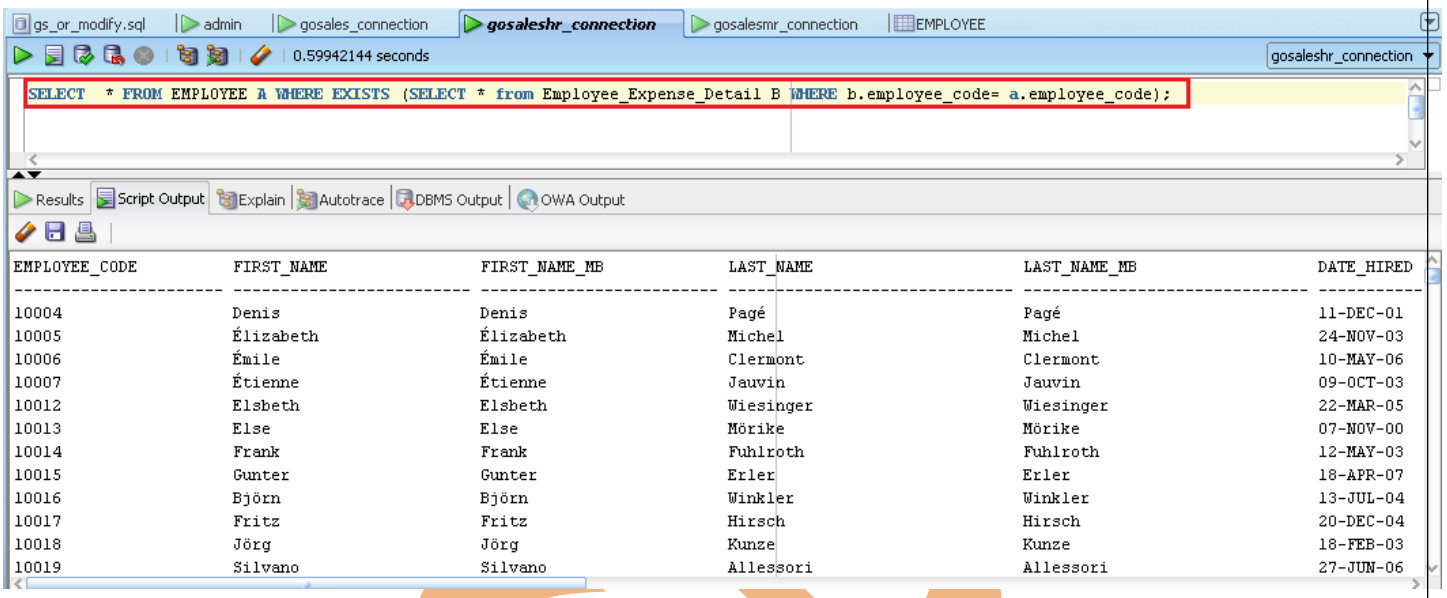
iii) Ensure that multiple sub-queries are in the most efficient order.

iv) Remember that rewriting a sub-query as a join can sometimes increase efficiency.

v) When doing multiple table joins consider the benefits/costs for each of EXISTS, IN, and table joins. Depending on your data one or another may be faster. Note: IN is usually the slowest.

For Example: Write the query as

*SELECT * from EMPLOYEE A WHERE EXISTS (SELECT * from Employee_Expense_Detail B WHERE b.employee_code= a.employee_code);*



EMPLOYEE_CODE	FIRST_NAME	FIRST_NAME_MB	LAST_NAME	LAST_NAME_MB	DATE_HIRED
10004	Denis	Denis	Pagé	Pagé	11-DEC-01
10005	Élizabeth	Élizabeth	Michel	Michel	24-NOV-03
10006	Émile	Émile	Clermont	Clermont	10-MAY-06
10007	Étienne	Étienne	Jauvin	Jauvin	09-OCT-03
10012	Elsbeth	Elsbeth	Wiesinger	Wiesinger	22-MAR-05
10013	Else	Else	Mörike	Mörike	07-NOV-00
10014	Frank	Frank	Fuhlroth	Fuhlroth	12-MAY-03
10015	Gunter	Gunter	Erler	Erler	18-APR-07
10016	Björn	Björn	Winkler	Winkler	13-JUL-04
10017	Fritz	Fritz	Hirsch	Hirsch	20-DEC-04
10018	Jörg	Jörg	Kunze	Kunze	18-FEB-03
10019	Silvano	Silvano	Allessori	Allessori	27-JUN-06

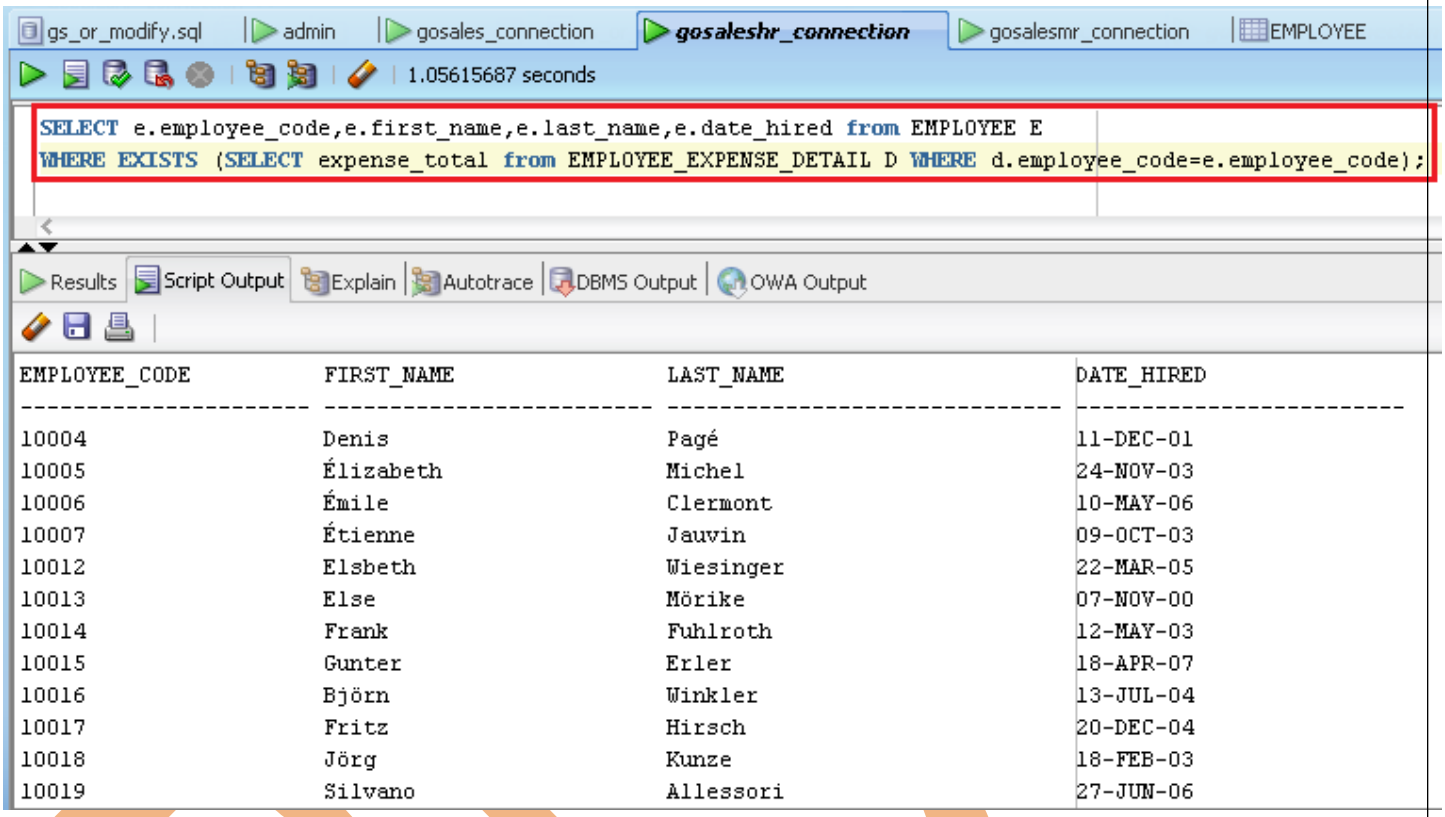
I**nstead** of :

*SELECT * from EMPLOYEE A WHERE employee_code IN(SELECT employee_code from Employee_Expense_Detail);*

vi)Use EXISTS instead of DISTINCT when using joins which involves tables having one-to-many relationship.

For Example: Write the query as

SELECT e.employee_code,e.first_name,e.last_name,e.date_hired from EMPLOYEE E WHERE EXISTS (SELECT expense_total from EMPLOYEE_EXPENSE_DETAIL D WHERE d.employee_code=e.employee_code);



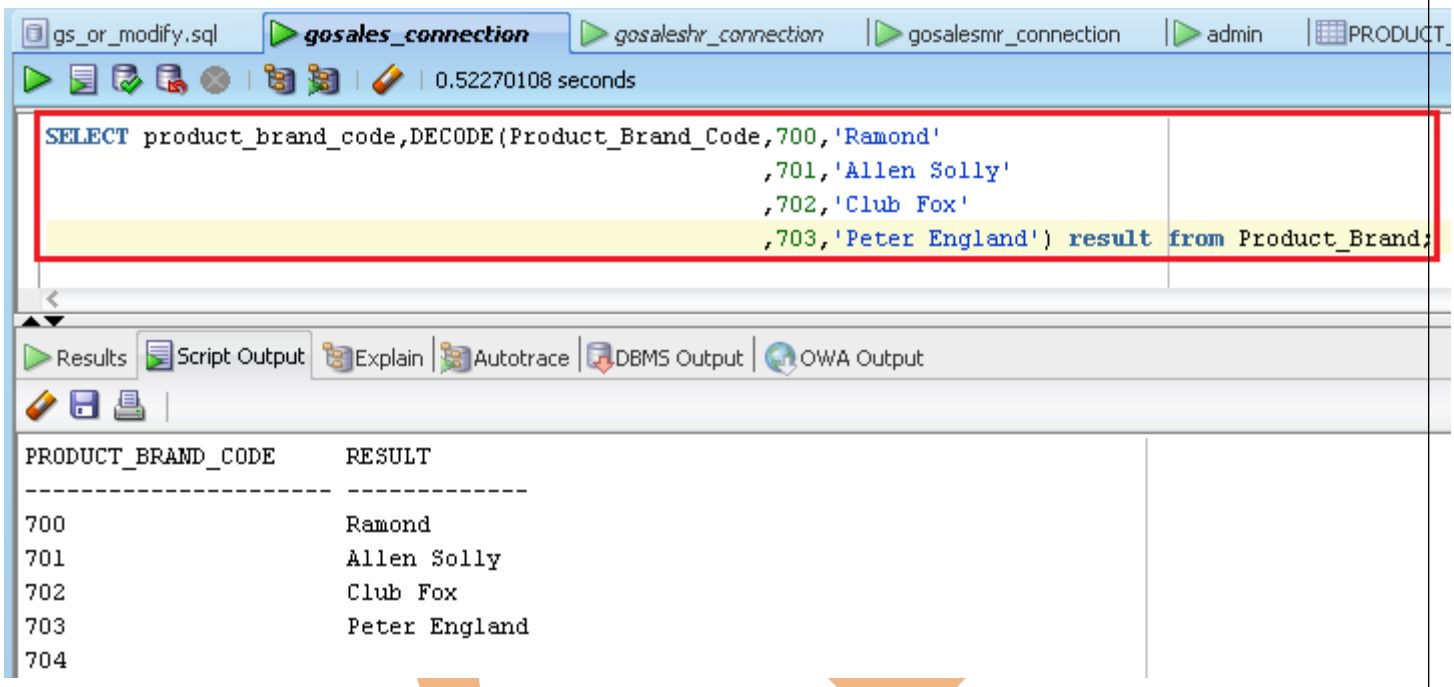
Instead of:

SELECT DISTINCT e.employee_code,e.first_name,e.last_name,e.date_hired from EMPLOYEE E,EMPLOYEE_EXPENSE_DETAIL D WHERE d.employee_code=e.employee_code;

Tuning Tips#11

Consider using DECODE to avoid having to scan the same rows repetitively or join the same table repetitively. Note, DECODE is not necessarily faster as it depends on your data and the complexity of the resulting query. Also, using DECODE requires you t change your code when new values are allowed in the field.

SELECT product_brand_code,DECODE(Product_Brand_Code,700,'Ramond' ,701,'Allen Solly' ,702,'Club Fox',703,'Peter England') result from Product_Brand;

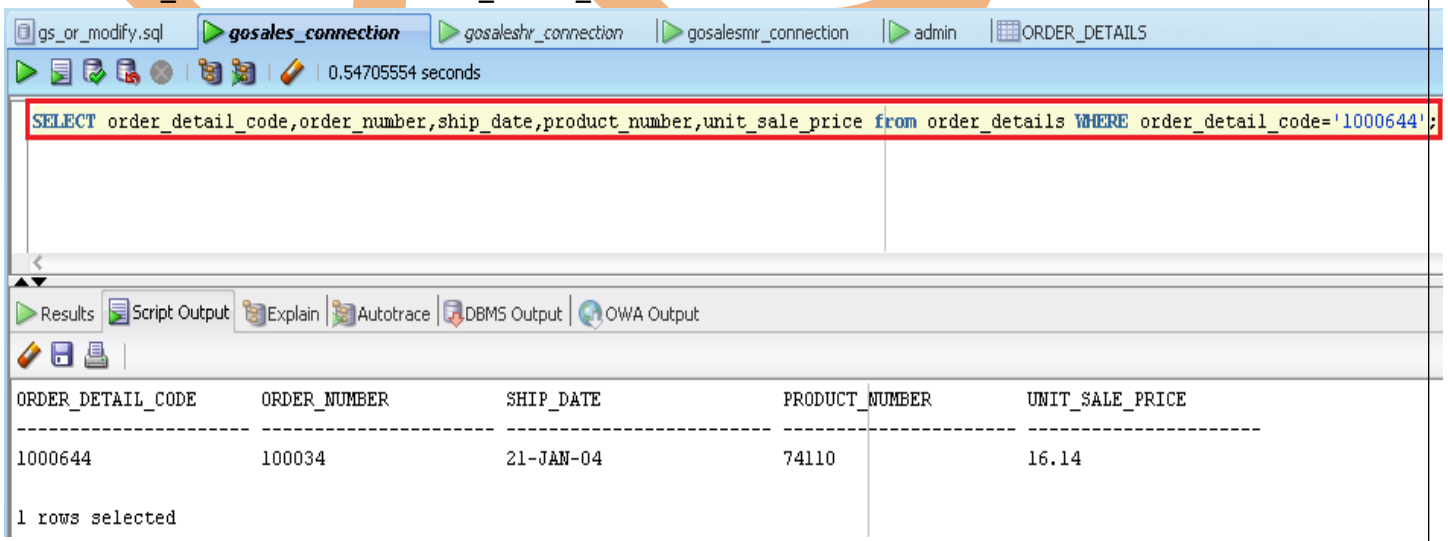


Tuning Tips#12

Oracle automatically performs simple column type conversions(or casting) when it compares columns of different types. Depending on the type of conversion, indexes may not be used. Make sure you declare your program variables as the same type as your Oracle columns, if the type is supported in the programming language you are using.

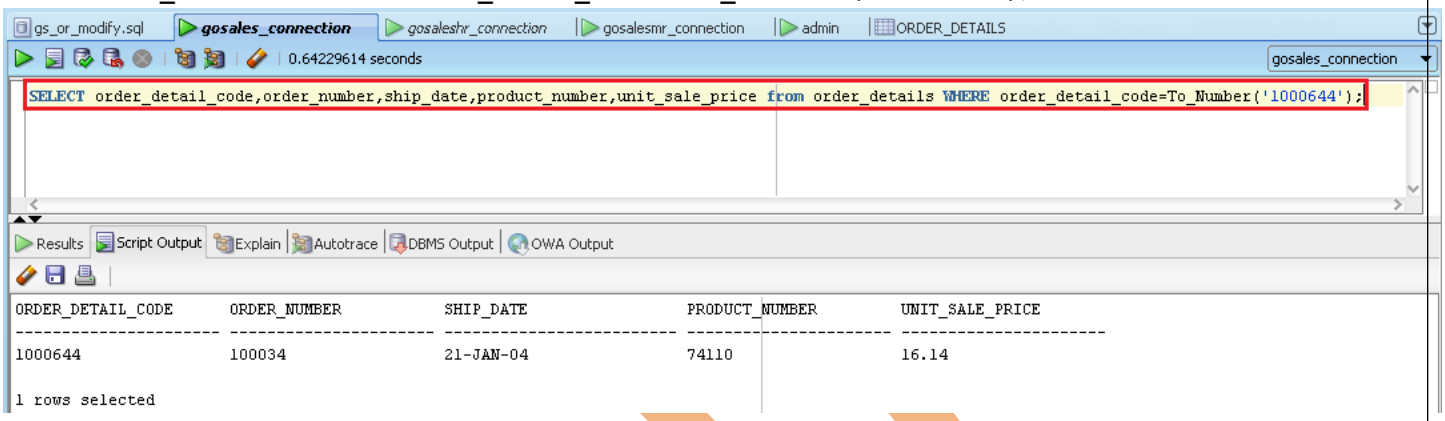
USE:

SELECT order_detail_code,order_number,ship_date,product_number,unit_sale_price from order_details WHERE order_detail_code='1000644';



HERE if order_detail_code indexed numeric, then after implicit conversion query will be:

SELECT order_detail_code,order_number,ship_date,product_number,unit_sale_price from order_details WHERE order_detail_code=To_Number('1000644');



Thus, index is used in this case.

Instead of:

SELECT order_detail_code,order_number,ship_date,product_number,unit_sale_price from order_details WHERE order_type=121;

HERE if order_type is indexed varchar2, then after implicit conversion query will be: SELECT emp_no, emp_name, sal FROM order_details WHERE TO_NUMBER(order_type) = 121;

Thus, index will not be used in this case.

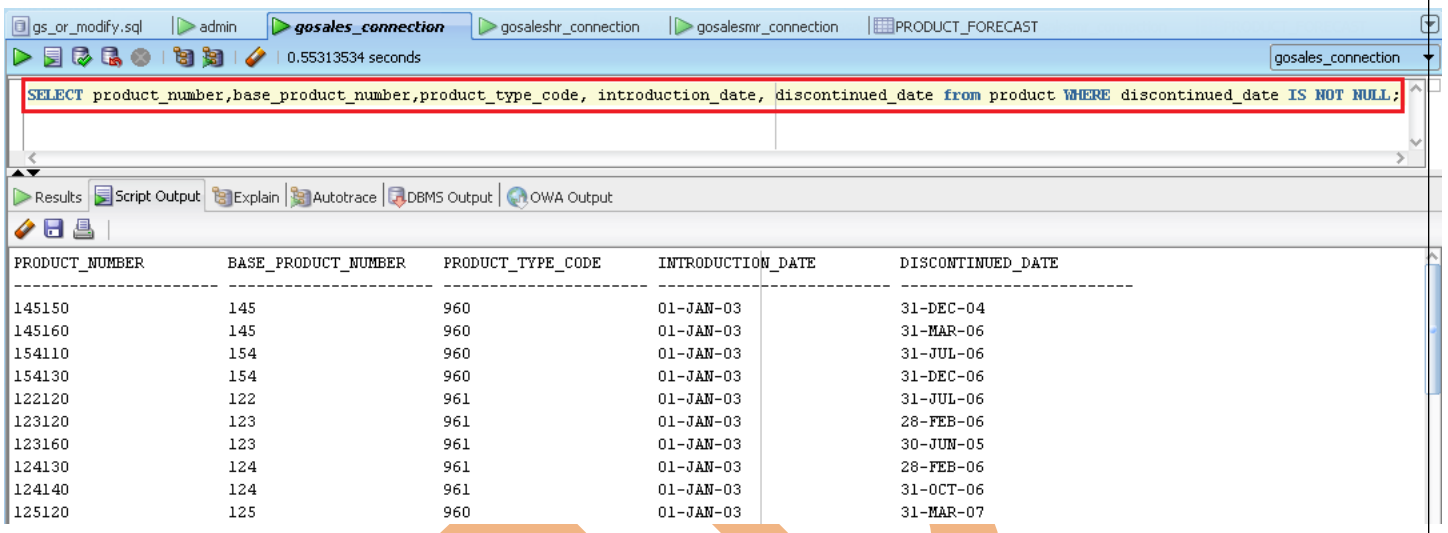
Tuning Tips#13

The most efficient method for storing large binary objects, i.e. multimedia objects, is to place them in the file system and place a pointer in the DB.

Tuning Tips#14

B-Tree Indexes do not store entries for NULL, so IS NULL is not indexable, but IS NOT NULL is indexable and thus if a huge table contains very few not null values then you should go for B-Tree indexes. On the other hand bitmap indexes support IS NULL condition.

SELECT product_number,base_product_number,product_type_code, introduction_date, discontinued_date from product WHERE discontinued_date IS NOT NULL;



Avoid using functions on indexed columns unless a function-based index is created; as it leads to full table scan even though index exists on the column.

Tuning Tips#15

Avoid using the following:
i)Boolean operators >, <, >=, <=, IS NULL, IS NOT NULL
ii)NOT IN, !=
iii)Like '%pattern', not exists
iv)Calculations on unindexed columns or (use union instead)
v)Having (use a WHERE clause instead when appropriate)

Tuning Tips#16

Do use the following:
i)Enable aliases to prefix all columns
ii)Place indexed columns higher in the WHERE clause
iii)Use SQL Joins instead of using sub-queries
iv)Make the table with the least number of rows, the driving table, by making it first in the FROM clause

Tuning Tips#17

Other important points for SQL Tuning

i)Establish a tuning environment that reflects your production database

ii)Establish performance expectations before you begin

iii)Always Design and develop with performance in mind

iv)Create Indexes to support selective WHERE clauses and join conditions

v)Use concatenated indexes where appropriate

vi)Consider indexing more than you think you should, to avoid table lookups

vii)Pick the best join method

viii)Nested loops joins are best for indexed joins of subsets

ix)Hash joins are usually the best choice for “big” joins

x)Pick the best join order

xi)Pick the best “driving” table

xii)Eliminate rows as early as possible in the join order

xiii)Use bind variables. Bind variables are key to application scalability

xiv)Use Oracle hints where appropriate

xv)Compare performance between alternative syntax for your SQL statement

xvi)Consider utilizing PL/SQL to overcome difficult SQL tuning issues

xvii)Consider using third party tools to make the job of SQL tuning easier

Performing these steps is easy and provides a tremendous benefit and performance boost. Follow these simple steps and you can increase your system performance.

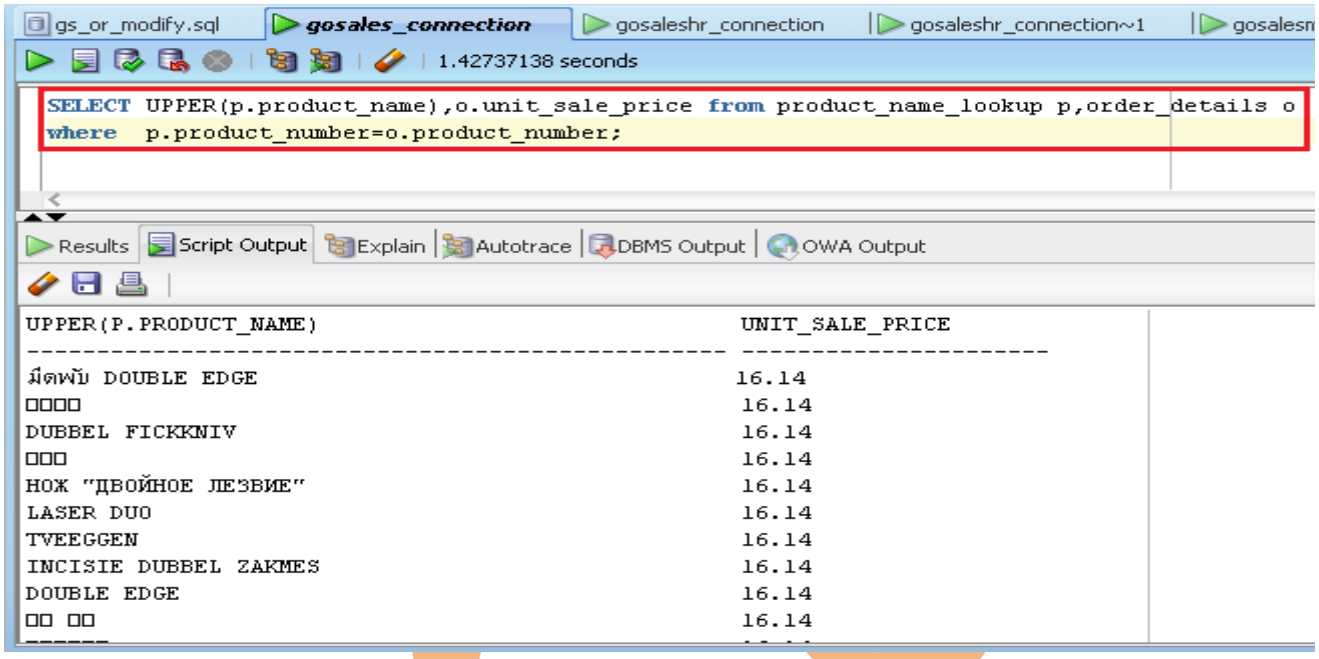
Tuning Tips#18

Using Function-based Indexes (FBI)

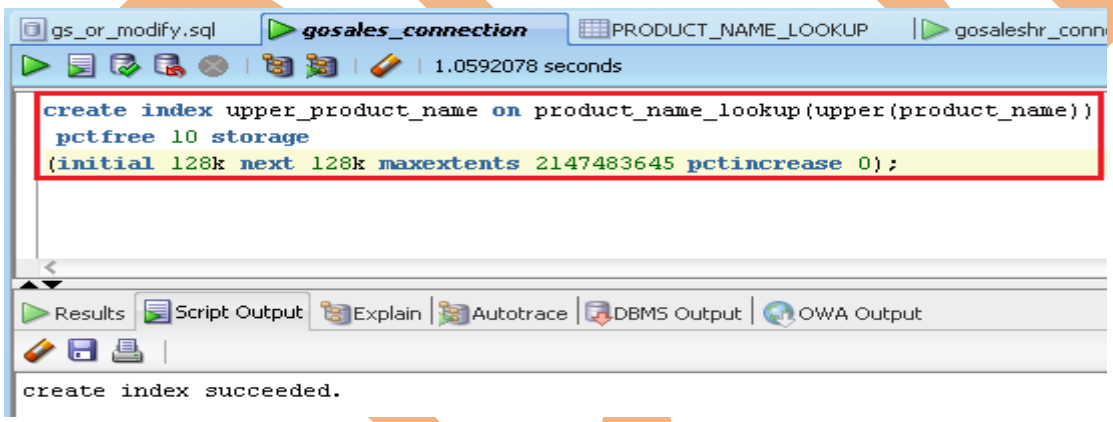
In almost all cases, the use of a built-in function like to_char, decode, substr, etc. in an SQL query may cause a full-table scan of the target table. To avoid this problem, many Oracle DBAs will create corresponding indexes that make use of function-based indexes. If a corresponding function-based index matches the built-in function of the query, Oracle will be able to service the query with an index range scan thereby avoiding a potentially expensive full-table scan.

The following is a simple example. Suppose the DBA has identified a SQL statement with hundreds of full-table scans against a large table with a built-in function (BIF) in the WHERE clause of the query. After examining the SQL, it is simple to see that it is accessing a customer by converting the customer name to uppercase using the upper BIF.

SELECT Upper(p.product_name),o.unit_sale_price from product_name_lookup p,order_details o WHERE p.product_number=o.product_number;



The table access full product option confirms that this BIF not using the existing index on the product_name column. Since a matching function-based index may change the execution plan, a function-based index can be added on upper(product_name).
Create index upper_product_name on product_name_lookup(upper(product_name)) pctfree 10 storage (initial 128k next 128k maxextents 2147483645 pctincrease 0);

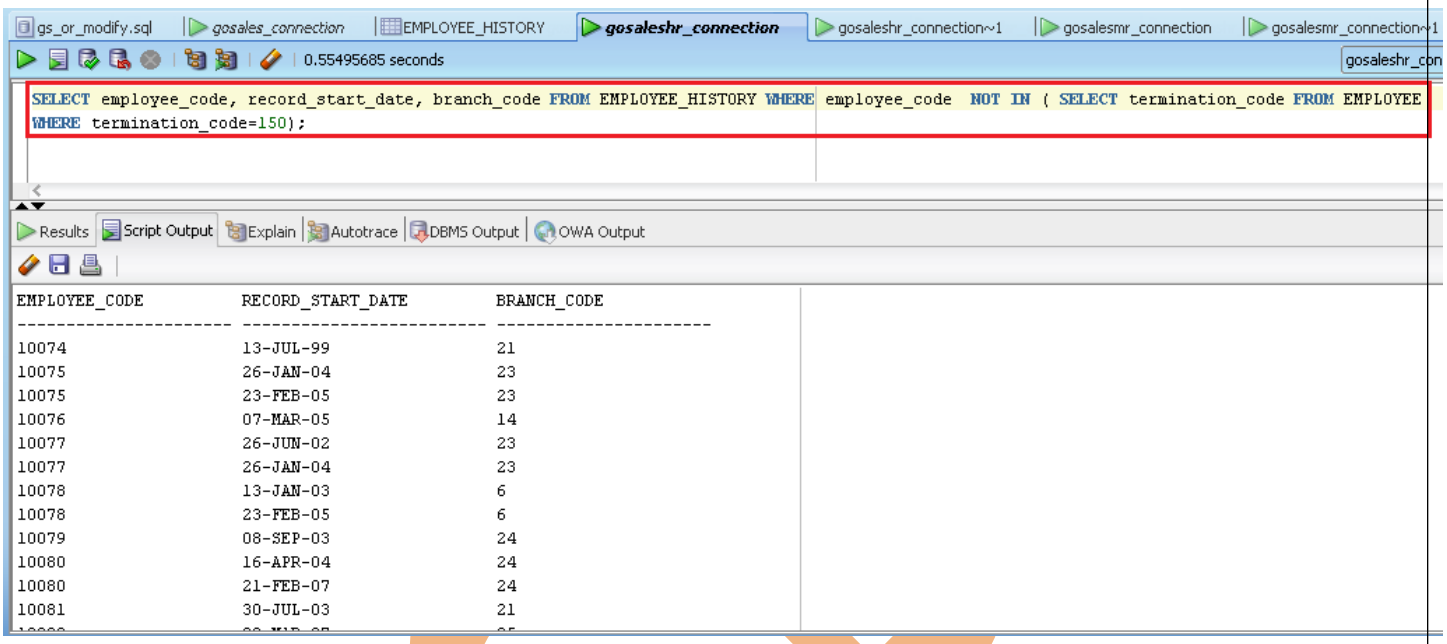


It can be risky to add indexes to a table because the execution plans of many queries may change as a result. This is not a problem with a function-based index because Oracle will only use this type of index when the query uses a matching BIF.

Tuning Tips#19

Using Temporary Tables

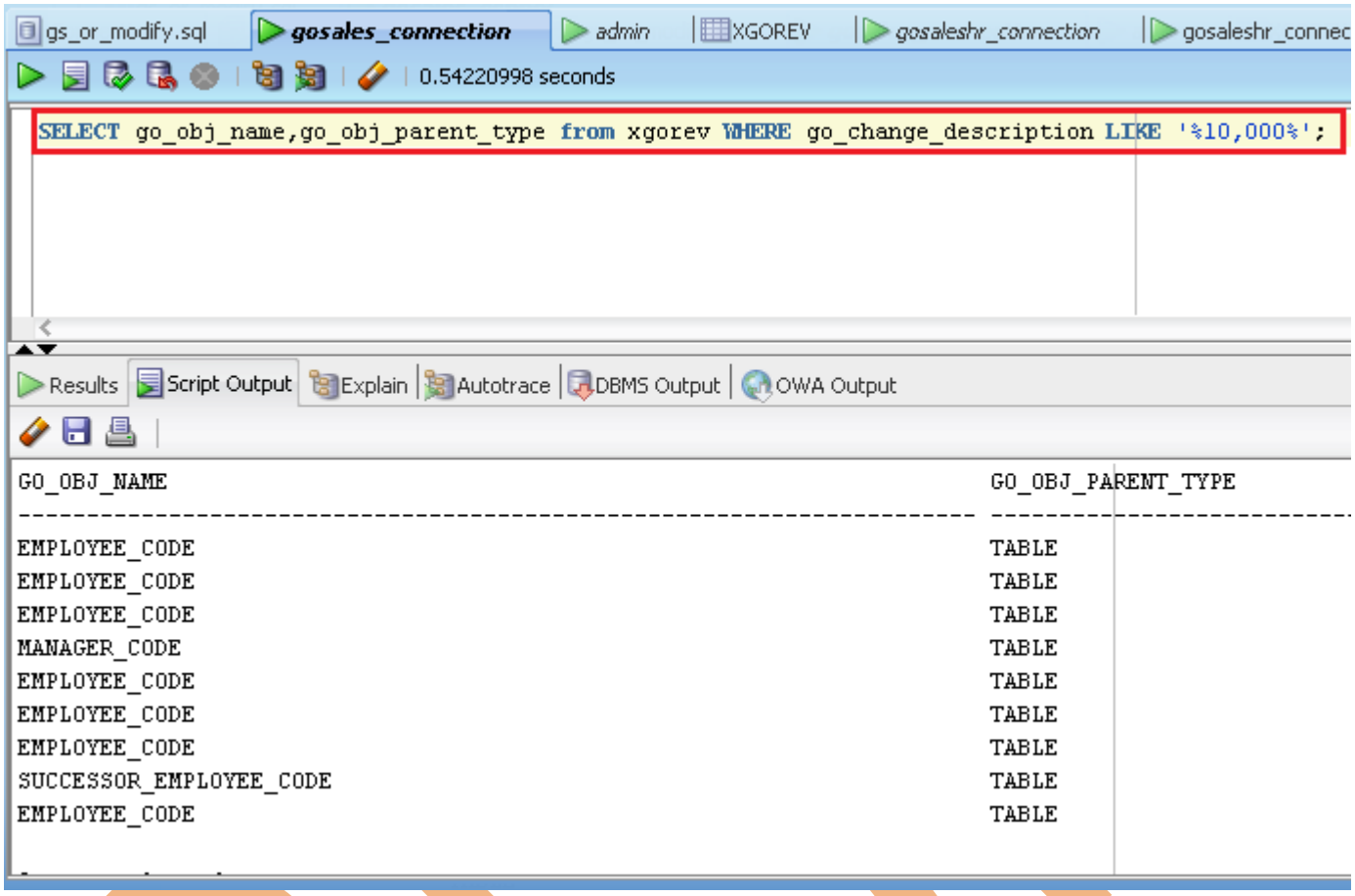
The prudent use of temporary tables can dramatically improve Oracle SQL performance. The following example from the DBA world can be used to illustrate this concept. The query could be formulated as an anti-join with a noncorrelated subquery as shown here:



Tuning Tips#20

Removing full-table scans with Oracle Text

One serious SQL performance problem occurs when the SQL LIKE operator is used to find a string within a large Oracle table column such as VARCHAR(2000), CLOB, or BLOB:



Since standard Oracle cannot index into a large column, their LIKE queries cause full-table scans, and Oracle must examine every row in the table, even when the result set is very small. The following problems can be caused by unnecessary full-table scans:

i) Large-table full-table scans increase the load on the disk I/O sub-system

ii)Small-table full-table scans(in the data buffer cause high consistent gets and drive up CPU consumption

The Oracle*Text utility, also called Oracle ConText and Oracle Intermedia, allows parsing through a large text column and index on the words within the column. Unlike ordinary b-tree or bitmap indexes, Oracle context ctxcat and ctxrule indexes are not updated as content is changed. Since most standard Oracle databases will use the ctxcat index with standard relational tables, the DBA must decide on a refresh interval. As a result, Oracle Text indexes are only useful for removing full-table scans when the tables are largely read-only and/or the end-users do not mind not having 100% search recall:

i) The target table is relatively static (e.g. nightly batch updates)

ii) The end-users would not mind missing the latest row data

Oracle Text works with traditional data columns as well as with MS-Word docs and Adobe PDF files that are stored within Oracle. Oracle Text has several index types:

iii) CTXCAT Indexes: A CTXCAT index is best for smaller text fragments that must be indexed along with other standard relational data (VARCHAR2).

WHERE CATSEARCH(text_column, 'ipod')> 0;

iv) CONTEXT Indexes: The CONTEXT index type is used to index large amounts of text such as Word, PDF, XML, HTML or plain text documents.

WHERE CONTAINS(test_column, 'ipod', 1) > 0

v)CTXRULE Indexes: A CTXRULE index can be used to build document classification applications.

These types of indexes allow users to replace the old-fashioned SQL LIKE syntax with CONTAINS or CATSEARCH SQL syntax.

When the query is executed with the new index, the full-table scan is replaced with a index scan, thereby greatly reducing execution speed and improving hardware stress:

Execution Plan

```
0      SELECT STATEMENT Optimizer=FIRST_ROWS
1    0   SORT (ORDER BY)
2    1     TABLE ACCESS (BY INDEX ROWID) OF 'BIGTAB'
3    2       DOMAIN INDEX OF 'TEXT-COLUMN_IDX'
```