Oracle SQL

Nassim Niclas Youssef | 08/04/2023 – Présent | Frisange (Domicile)- Luxembourg

Table of Contents

Basic SQL

Select Statement

Select * from table schema.table1 t1; => the * mean select all the columns in this table, and the t1 is simply as short name for our table, P. S here we can use the short name of the table to refer to a column like: **select t1.col1 from sch1.table t1 .**

To use a different column name for our result we can : **select t1.col1 as 'Column 1'** or simply without the **as** keyword

Sorting

Select ..... from table ..... order by col-name desc,col-name2 => the desc means that we are sorting the results by descending order of the **col_name** and then if we find 2 equal value we will resolve by another sorting with respect to **col-name2** but ascending.

Filtering

Select .... From .... Where tablealias.col1 = value => it's a basic filtering expression using where, here we are filtering the results that have col1 equals to value

General form:

Select .... From .... Where <expression> <operator> <expression> => Expression like col1 =1 or col1 > value2. Operator can be (**or,and and so on...**) **. Example: Select col1 from table1 where col1 = 'test' and col2 < 3**

Substitution Variables

This kind of operation are only meant for SQL Plus

define var_name = value; => here we are defining a variable

select * from xyz where col1 = &var_name => here we are just using the variable we just created in our query

undefine var_name ;=> simply is to delete the variable

Set verify off ; simply when we are executing a querry with a defined variable by default SQL Plus will show the query text with the value of the defined variable. Example:

SET VERIFY OFF; DEFINE X = 10 ; SELECT .... Where col1 = & x will show nothing on the terminal

SET VERIFY ON; DEFINE X = 10 ; SELECT .... Where col1 = & x will select .... Where col1 = 10. P.S: **Select ... from where col = &&x; if x variable is undefined it will ask us to put in the console**

ROW Limiting

Fetch next n cols

1-SELECT *
2-FROM MIO_RES_COMM_LBP_M2 mrclm
3-FETCH NEXT 3 ROWS ONLY

What we are doing here is basically after fetching the result from ligne 1 to ligne 2 we are taking the first 3 rows. P.S: if before the Fetch statement we have an order by will take the first 3 results by the order by statement

Fetching Next N rows after skipping M rows

1-SELECT *
2-FROM MIO_RES_COMM_LBP_M2 mrclm
3- OFFSET <M> Rows
4-FETCH NEXT N ROWS ONLY

Fetching Top N % row

1-SELECT *
2-FROM MIO_RES_COMM_LBP_M2 mrclm
3-FETCH NEXT <N> PERCENT ROWS ONLY

Notes:

If we put **with ties** instead of only we will fetch the top rows and also we also add any row that have ties (equal values) to the ton N rows

Operator

Comparison Operators

Operator	Description
=	equal
!= or <>	Not equal to
>,<,=<,=>	Less than, greater than and so on
in	Equal to any value in a list
Not in	Opposite of in
Between x and y	Value between x and y, >=x and <=y
Is null	Value is null
Is not null	Opposite of null

Usage: **select … from …. Where col1 between value1 and value2**

How to concatenate

Select col1 || col2 as concat_col from …

Wild card search

Select .... From .... Where col1 like <pattern> => we are searching for a value that match a pattern

Select .... From .... Where col1 not like <pattern> => we are searching for a value that doesn't match a pattern

keyword	Description
%	Any string
_	Any single char

Example of patterns

Col1 like %A => matching col1 with anything with the last char equal to A

Col1 like A% => matching col1 with anything with the first char equal to A

Col1 like _A% => matching col1 with any string that begin with single char followed by the letter A and then followed by any string for example: **RA**acasdar132er

Col1 like A_B% => matching col1 with anything with the first char equal to A followed by any single char then followed by B and finally any string like : AcBafr32r

Col1 like '%_%' escape '\' : any string char followed by the _ char then any string. P.S: the escape keyword is used to say that when we find the escape char next to character, use this character as value even if it's a keyword. So basically, when we need to match a pattern with _ or % character it should be preceded by the escape char

Logical Operator

Operator	Description
Not	If false then evaluate to true , if true evaluate to false
And	Expression and expression should both be true to evaluate the entire and expression as true
Or	It's like and but one at least should evaluate true

Example:

Select ... from dual where 1 = 1 and 2 =2 => we have a match
Select ... from dual where 1 = 1 and 2 !=2 => we don't have a match
Select ... from dual where 1 = 1 or 2 !=2 => we a match
Select ... from dual where 1 != 1 or 2 !=2 => we don't have match

Set operators (Union, Intersect, Except, Minus)

Union

The below query will return rows with column of COL1,COL2 from 2 table and the rows have distinct value

Select COL1,COL2 from table1
Union
Select COL1,COL2 from table2

Same as above but we will return everything even the duplicated

Select COL1,COL2 from table1
Union all
Select COL1,COL2 from table2

Intersect

The below query will return all distinct intersection between table1 and table2

```
SELECT REQUEST_PATTERN
FROM MIO_CONFIG_SECURITY table1
INTERSECT
SELECT REQUEST_PATTERN
FROM MIO_CONFIG_SECURITY table2
```

Minus (Or the except of SAS)

The below will get all results of table1 without the one that are also in common with table2

```
SELECT REQUEST_PATTERN
FROM MIO_CONFIG_SECURITY table1
MINUS
SELECT REQUEST_PATTERN
FROM MIO_CONFIG_SECURITY table2
```

Single Row Functions

Numeric Function

ABS(NUM) : the absolute value

FLOOR(num): returns 10 if 10.4 and returns 11 if num = 10.5 or 10.6 and ....

CEIL(num): return 10 if num=10.4 or 10.5 or 10.6

MOD(X,Y): x%y

POWER(X,Y) : x to the power of y: X^Y

SIGN(X): -1 if X is negative, 1 if positive and 0 if X = 0

TRUNC(N1,N2): keep only N2 digit after the decimal, 10.44543,2 => 10.44

WIDTH_BUCKET(Val,min,max,max_num_of_tranches): this function will determine in which tranche the val belong, We need to specify the min val and the max val, and the number of tranches example:

SELECT WIDTH_BUCKET (6.4, 0,10,5) **FROM** DUAL; [0-2[=> tranche = 1 ; [2,4[=> tranche 2; [4,6[tranche 3; [6,8[tranche 4, [8,10[tranche 5, [10, infinity[tranche 6

Round(X,Y): round the number to Y position after the decimal point
Example: 10.16 ; 1 => 10.2 15.434;-1 => 20 we are based on the five so the 1 will rounded to 2 10.16;0 => 10 because .16 is less than 0.5

String Function

Character Functions	Description
CONCAT(str1, str2)	It concatenate 2 strings
REPLACE(str,'x','y')	It replaces x character in the string "str" to y
SUBSTR(str , 1,10)	It read the characters from position 1 to 10
UPPER(str)	Make the string to upper case
LOWER(str)	Make the string to lower case
INIT(str)	Make the string to init char. Example world => World
LPAD(str , 10, '*')	Left padding of string with "*" till length 10
RPAD(str , 10, '*')	Right padding of string with "*" till length 10
LTRIM(str ,'^')	Left Trim the string with "^"
RTRIM(str ,'^')	Right Trim the string with "^"
TRANSLATE(str ,'xyx',pqr')	Does character by character replace as per sequence provided

INIT should be INITCAP : INITCAP('test test') => Test Test; will capitalize the first char of each word

Translate(str,seq1,seq2): TRANSLATE('123ABC', '1A','4Z') => 423ZBC basically we will replace char of seq1@i with seq2@i in the original string, in this case 1 will become 4 and A will become Z

Date Function

Date Functions	Description
SYSDATE	Return the current date and time
SYSTIMESTAMP	Return the current date and time up to microseconds
ADD_MONTHS(date, n)	It adds n months to a date
MONTHS_BETWEEN(date1, date2)	Find the months between two dates. Date1 should be greater else you get negative value
NEW_TIME	NEW_TIME(date, from_timezone, to_timezone)
NEXT_DAY(SYSDATE,'MONDAY')	Next day after a particular date i.e.- SYSDATE
LAST_DAY(date)	Last date of the month
TO_DATE(str)	Convert a string to a Date
TO_TIMESTAMP	Convert a string to a Timestamp
EXTRACT (datetime)	Extract date, Month & year from a date
ROUND (date, 'DAY')	Round a date by DAY, MONTH, Q=Quarter.
TRUNC (date, 'DAY')	Truncate a date by DAY, MONTH, Q=Quarter.
TO_CHAR (date)	Convert date/Timestamp to Character

SELECT SYSDATE From DUAL => return a date in a readable form

SELECT SYSTIMESTAMP From dual => return a date in a readable form with micro seconds

SELECT ADD_MONTHS(SYSDATE,2) => add 2 months to a date

EXTRACT(<Date time component> from date_type_value) => example : EXTRACT(DAY from SYSDATE), So basically we treat the date_value as a table and in our extract we write a sql to select the column but without the select keyword

Type Conversion

TO_CHAR(date,format): TO_CHAR(SYSDATE,'DAY') will return the day name (Monday , Tuesday ....) or we can simply convert a number to string
TO_DATE(val,format): converting a string to date object example: TO_DATE('31-12-2023','dd-MM-YYYY)
TO_TIMESTAMP(val,format) : same as TO_DATE but with microseconds mention
TO_NUMBER(str) => convert a string to numeric type

Null Functions

Character Functions	Description
COALESCE	COALESCE returns the first non-null *expression* in the expression list
NULLIF(expr1, expr2)	WHEN expr1 = expr2 THEN NULL ELSE expr1 END
NVL(expr1, expr2)	If *expr1* is null, then NVL returns *expr2*
NVL2(expr1, expr2, expr3)	If *expr1* is not null, then NVL2 returns *expr2*. If *expr1* is null, then NVL2 returns *expr3*.

COALESCE(str1,str2,....) or COALESCE(str[]): return the first non null value

If Else Functions

Decode(expr,val1,out1,val2,out2,.....,out)

Basically it like a switch case:

If expr = val1 then return out1, if express=val[i] return out[i] if not match return out

P.S: the number of val@i and out@i should be even because the last arg is always the default

Aggregation and GroupBy Function

Where vs Having

Where is used to filter data before aggregation

Having used to filter data after aggregation

Select * from table groupby .... Having col1 = value1

Aggregate Function

First we need to include group by like this

Select ... from table groupby col1,col2....

P.S: the select should have aggregation function since we are only allow to return one row per group in this case for the exception of the columns in the groupby they can be selected since we are sure that their unique to each group. Some other case like CUBE and ROLLUP have different approaches

A list of aggregation function is available at:

https://docs.oracle.com/database/121/SQLRF/functions003.htm#SQLRF20035

Pivot and Unpivoting

As you can see from the output, the number of pivot columns is doubled, combining `category_name` with `orders` and `sales` .

Finally, use `status` as the pivot columns and `category_name` as rows:

```
SELECT * FROM order_stats
PIVOT(
    COUNT(order_id) orders,
    SUM(order_value) sales
    FOR status
    IN (
        'Canceled' Canceled,
        'Pending' Pending,
        'Shipped' Shipped
    )
)
ORDER BY category_name;
```

The following picture shows the output:

CATEGORY_NAME	CANCELED_ORDERS	CANCELED_SALES	PENDING_ORDERS	PENDING_SALES	SHIPPED_ORDERS	SHIPPED_SALES
CPU	13	4122040.7	12	3647918.1	51	12272034.23
Mother Board	12	679121.39	12	872381.21	53	3366915.55
Storage	14	3023747.6	16	2271252.28	63	8699160.96
Video Card	9	1677597.4	11	3040231.59	39	9069691.63

The count and the Sum are aggregation function that represent the column that will be put under the pivoted column, and the for(status) is to say that the status column will be the one that will rotate , the in statement is specify what are the values that will be considered in the status to make this rotation (simply we are rotating for specific values of status)

The Unpivot is like the pivot but instead of aggregation function we specify the name of new column that will contain the value after rotation.

Near the FOR we put the column name that will contain the values of column name and in the IN statement we are simply specifying what are the column name that will rotated and to what value they point to

ID	FISCAL_YEAR	PRODUCT_A	PRODUCT_B	PRODUCT_C
1	2017	(null)	200	300
2	2018	150	(null)	250
3	2019	150	220	(null)

Oracle UNPIVOT examples

This statement uses the `UNPIVOT` clause to rotate columns `product_a`, `product_b`, and `product_c` into rows:

```
SELECT * FROM sale_stats
UNPIVOT(
    quantity  -- unpivot_clause
    FOR product_code --  unpivot_for_clause
    IN ( -- unpivot_in_clause
        product_a AS 'A',
        product_b AS 'B',
        product_c AS 'C'
    )
);
```

The following picture shows the output:

ID	FISCAL_YEAR	PRODUCT_CODE	QUANTITY
1	2017	B	200
1	2017	C	300
2	2018	A	150
2	2018	C	250
3	2019	A	150
3	2019	B	220

Analytical Functions (Windowing Function)

Simply those function are used to calculate a specific aggregate function but without the groupby functions it's like we calculating the normalized value of a column with respect to a sum calculated by specific group

Select sum(col1) OVER (Partition by col2 order by col3) from table

Ordinary Aggregation Function

Select MAX(col1) over (Partition By col2) from table1

Here we are selecting the max relatively to a group

LISTAGG

Is a function to create a string by aggregating multiple strings from specific group specified by the group by

SELECT LISTAGG(col1,sperator) WITHIN GROUP (ORDER BY COLS) From T groupby cols

Here we are concatenating all the value of col1 with respect to the cols in groupby

DENSE_RANK

Is a ranking function to compute the rank of a row respect to a group

SELECT DENSE_RANK() OVER (PARTITION BY cols order by other_cols)

Partition by is specify the rank relative to what group, the order by is crucial because we will compare the original row with the ones that will created in the OVER statement

Keep Statement

Select AVG(COL) KEEP (DENSE_RANK FIRST|LAST ORDER BY COL1) OVER (PARTITION BY COL2) => here we are calculating the average of COL over a partition by COL2 and we are only taking the values that are at the first rank (as per specified by the keep statement and the dense rank first)

Group By with subtotals

Grouping SET

GROUP BY GROUPING SET(A,B,C) is equivalent to GROUP BY(A) UNION GROUP BY B UNION GROUP BY C

Example: Select sum(COL) ... GROUP BY GROUPING SET ((A,B,C), (A,B) , (A)) this kind will produce total and subtotal for each of A,B,C group the totals of each A,B group and then the total of each A group

Roll UP

GROUP BY ROLLUP(A,B,C) => GROUP BY GROUPING SET ((A,B,C), (A,B) , (A), ())

CUBE

CUBE(A,B,C) => GROUPING SET (A,B,C),(A,B),(A,C),(B,C),(A),(B),(C)

Composite Columns

ROLLUP (A , (B,C) ,D) => here (B,C) is 2 columns but treated as one => GROUPING SET ((A , (B,C) , D) , (A, (B,C)) , (A) , ()) => as we can see we either have B,C together or any of them

Multiple Expression

GROUP BY COL <ROLLUP or Cube>(COL2,COL3...) => basically we are executing the rollup or cube for each of COL Group, Example: COL have values(1,2,3) the RollUP(COL2,COL3,...) will be executed 3 time for value 1,2,3

Grouping or Summary column with no null valuesSo

SELECT GROUPING(COL1),GROUPING(COL2) FROM T GROUP BY ROLLUP(COL1,COL2) => Basically when we do rollup and when we select the COLs of the Rollup and when we reach the subtotal some column of the rollup will be null in the final result so the grouping function will return 0 or 1 if the col of the rollup is actually representing a subtotal or not

GROUPING_ID

Grouping_ID(COL1,COL2) => fromBinaryToInteger(GROUPING(COL1) concat GROUPING(COL2))

Group_ID

Group_ID() is simply returning the ROW_ID relative to the group

Lead and LAG

Lead and Lag take 3 args:

Colsname,number of steps in forward(lead) or backward in case of lag, default

Basically the lead will go n steps in forward (n rows from the current position)

Basically the lag will go n steps in backward (n rows from the current position)

P.S: the lead and lag requires the OVER keyword

With Clause (Recursive and Non-recursive)

Query type

Select ... from table where exists (subquery) => means if the subquery return at least one result the evaluation is true. Not exists follows the same logic but to be evaluated to true it the subquery should return zero queries

Query Types	Description
Subquery (Subquery will be found in the WHERE clause)	SELECT * FROM emp WHERE deptno IN (SELECT deptno in dept)
Co-related Sub query (For each record is outer query Inner query executes based open conditions)	SELECT * FROM emp e WHERE deptno= (SELECT deptno FROM dept d WHERE e.deptno = d.deptno)
Inline Views (Subquery in FROM clause	SELECT * FROM (SELECT * FROM emp WHERE deptno=10)

Joins

Ordinary Joins

Inner join

Select ... from table1 Inner join table2 on table1.col = table2.col

Left/Right/Full Join

Select ... from table1 Left outer join table2 on table1.col = table2.col

Select ... from table1 Right outer join table2 on table1.col = table2.col

Select ... from table1 FULL outer join table2 on table1.col = table2.col

Cartesian Join

Select .... From table1,table2 => the following will produce a cartesian join P.S: if we put a where we will be producing a filtered join equivalent to left join

Filtered Cartesian Join

Select .... From table1,table2 where condition => here we are doing a filtered cartesian join that is similar in results with a left join but not similar in execution and internal implementation

Table

Create Table

Create table <table-name> (
<col_name> <col_type>,
<col_name> <col_type>)
Or
Create table <table_name> AS (SQL)

Some Special Tables

Table

DATA DICTIONARY

```
SELECT * FROM all_Objects
SELECT * FROM all_tables, user_tables
SELECT * FROM user_tab_cols

COMMENT ON COLUMN product.product_id IS 'Unique ID for a Product';
COMMENT ON COLUMN product.product_code IS 'Unique code for a Product';
```

DATA DICTIONARY

```
SELECT * FROM ALL_TAB_COMMENTS
SELECT * FROM USER_COL_COMMENTS
```

USER_SEGMENTS,
COMMENT ON COLUMN <table-name>.<col_name> is str_value

Temporary Table

Temporary Tables

Operation	Details
CREATE GLOBAL TEMPORARY TABLE <table name> (id NUMBER, description VARCHAR2(20)) ON COMMIT DELETE ROWS;	Delete rows after commit.
CREATE GLOBAL TEMPORARY TABLE <table name> (id NUMBER, description VARCHAR2(20)) ON COMMIT PRESERVE ROWS;	Keep the rows after commit.

Temporary table can be created to be used by multiple session, but only the session that puts the data can see this data. Example: session1 create table1, session2 can see it, session1 insert data1 into this table only session1 will see data1, session2 cannot see anything but if session2 put data2 into this table session2 will only data2 session1 cannot see data2

Table With Virtual Column (Generated Columns)

This type of column is not stored on the disk but it's generated computed

```
create table t1 (
  id              number,
  product         varchar2(50),
  price           number(10,2),
  price_with_tax  number(10,2) generated always as (round(price*1.2,2))
virtual
);
```

So in our insert we don't need to mention it

External Table

Simply here we are able to use our filesystem to create or read table that can be different than the file system

Step-1 Creation Of Directory

Create Directory <dir_name> AS <path>

Step2- Grant the permissions

Grant Read,Write on directory <dir_name> To <user>

Step3 Create

CREATE TABLE <table_name> (<some_col_definitions>)
ORGANIZATION EXTERNAL(
TYPE oracle_loader
Default directory <dir_name>
ACCESS PARAMETERS (FIELDS TERMINATED BY <delimiter>)
LOCATION(<csv_path>)
);
PS: We should always create the csv file in the directory dir

Step4

We can simply edit the CSV file and the new edits or adds will visible

Step5

The steps above are simply to associate an external file to table, so here we can use normal sql to query data from those external file

Data Type

Datatype	Description	Column Length and Default
CHAR (*size*)	Fixed-length character data of length *size* bytes.	Fixed for every row in the table (with trailing blanks); maximum size is 2000 bytes per row, default size is 1 byte per row. Consider the character set (one-byte or multibyte) before setting *size.*
VARCHAR2 (*size*)	Variable-length character data.	Variable for each row, up to 4000 bytes per row. Consider the character set (one-byte or multibyte) before setting *size.* A maximum *size* must be specified.
NCHAR(*size*)	Fixed-length character data of length *size* characters or bytes, depending on the national character set.	Fixed for every row in the table (with trailing blanks). Column *size* is the number of characters for a fixed-width national character set or the number of bytes for a varying-width national character set. Maximum *size* is determined by the number of bytes required to store one character, with an upper limit of 2000 bytes per row. Default is 1 character or 1 byte, depending on the character set.
NVARCHAR2 (*size*)	Variable-length character data of length *size* characters or bytes, depending on national character set. A maximum *size* must be specified.	Variable for each row. Column *size* is the number of characters for a fixed-width national character set or the number of bytes for a varying-width national character set. Maximum *size* is determined by the number of bytes required to store one character, with an upper limit of 4000 bytes per row. Default is 1 character or 1 byte, depending on the character set.
CLOB	Single-byte character data.	Up to 2^32 - 1 bytes, or 4 gigabytes.
`NCLOB`	Single-byte or fixed-length multibyte national character set (NCHAR) data.	Up to 2^32 - 1 bytes, or 4 gigabytes.
`LONG`	Variable-length character data.	Variable for each row in the table, up to 2^31 - 1 bytes, or 2 gigabytes, per row. Provided for backward compatibility.

NUMBER (p, s)	Variable-length numeric data. Maximum precision *p* and/or scale *s* is 38.	Variable for each row. The maximum space required for a given column is 21 bytes per row.
DATE	Fixed-length date and time data, ranging from Jan. 1, 4712 B.C.E. to Dec. 31, 4712 C.E.	Fixed at 7 bytes for each row in the table. Default format is a string (such as DD-MON-YY) specified by NLS_DATE_FORMAT parameter.
BLOB	Unstructured binary data.	Up to 2^32 - 1 bytes, or 4 gigabytes.
BFILE	Binary data stored in an external file.	Up to 2^32 - 1 bytes, or 4 gigabytes.
RAW (size)	Variable-length raw binary data.	Variable for each row in the table, up to 2000 bytes per row. A maximum *size* must be specified. Provided for backward compatibility.
LONG RAW	Variable-length raw binary data.	Variable for each row in the table, up to 2^31 - 1 bytes, or 2 gigabytes, per row. Provided for backward compatibility.
ROWID	Binary data representing row addresses.	Fixed at 10 bytes (extended ROWID) or 6 bytes (restricted ROWID) for each row in the table.
MLSLABEL	Trusted Oracle datatype.	See the *Trusted Oracle* documentation.

View

View is simply representation of some dataset from one or multiple tables, View is not a physical table but like a stored query. Example when select * from view , what we are actual doing is executing a query to get the result that this view represent and then using the result as a temporary table for an outer query. View doesn't use any physical space it's just a query save in our database. View can also be used for security meaning giving the user limited access to the table data with limited privileges

Create

Create or replace view <view_name> as (SQL querry)

Drop

Drop view <view_name>

Querying

Select ... from <view_name> ....

Key-Preserved Tables in a view

In simple terms, a table is key preserved if the table key participates in the view as a key. In short, a key-preserved table has its key columns preserved through a SQL join. Means if all the key columns are still present or if the rows in the views can refer to one and only one row from original tables (One Row from the view should exactly match one row from the original table(s), in this case a insert or delete operation is allowed

Constrains And Column Def modifications

Operation	Details
NOT NULL	prohibits a database value from being null
UNIQUE	prohibits multiple rows from having the same value in the same column
PRIMARY KEY	combines a NOT NULL constraint and a unique constraint in a single declaration
FOREIGN KEY	requires values in one table to match values in another table
CHECK	requires a value in the database to comply with a specified condition

Constrains Creation on Table Creation

NOT NULL

CREATE TABLE <table_name> (XYZ NUMBER NOT NULL, ...)

Unique

```
CREATE TABLE table_name
(
 column1 datatype [ NULL | NOT NULL ],
```

```
  column2 datatype [ NULL | NOT NULL ],
  ...

  CONSTRAINT constraint_name UNIQUE (uc_col1, uc_col2, ... uc_col_n)
);
```

Primary Key

```
CREATE TABLE table_name
(
  column1 datatype [ NULL | NOT NULL ],
  column2 datatype [ NULL | NOT NULL ],
  ...
  CONSTRAINT constraint_name PRIMARY KEY (uc_col1, uc_col2, ... uc_col_n) );
```

Foreign Key

```
CREATE TABLE table_name
(
  column1 datatype [ NULL | NOT NULL ],
  column2 datatype [ NULL | NOT NULL ],
  ...
  CONSTRAINT constraint_name FOREIGN  KEY (col1,col2,col3) REFERENCES <parent_table>
(col_p_1,col_p2,col_p_3 ) );
```

Col1,Col2,Col3 is from table_name
Col_p_1,col_p_2,col_p_3 is from the parent table

Check

```
CREATE TABLE table_name
(
  column1 datatype [ NULL | NOT NULL ],
  column2 datatype [ NULL | NOT NULL ],
  ...
  CONSTRAINT constraint_name CHECK (col1 = something and col2 not equal to 4) )
[Disable];
```

In the check we are simply writing a normal where, P.S the disable is optional

Modify/ADD

Constraints

Alter table table_name <ADD Constraint or Modify> <constraint_name> constraint definition

Example:

Alter table t1 ADD Constraint <constaint_name> Primary Key (...)

ALTER TABLE T MODIFY <constraint_name> PRIMARY KEY (...)

ALTER TABLE suppliers DISABLE CONSTRAINT check_supplier_id;

Columns

Alter Table t add <col_name> <col def>
Alter Table t modify COL_NAME <col_def>
Examples:

Alter table t ADD COL_1 NUMBER(7,2) NOT NULL
ALTER TABLE t MODIFY COL_1 NUMBER(10,2)

Virtual Column

```
alter table t1 modify (
  price_with_tax number(10,2) as (round(price*1.3,2))
);
```

```
alter table t1 add (
  price_with_tax number(10,2) as (round(price*1.2,2))
);
```

Drop and Disable constraints

```
ALTER TABLE emp DROP CONSTRAINT FK_DEPTNO
ALTER TABLE emp DISABLE CONSTRAINT FK_DEPTNO;
```

P.S the keyword DISABLE can be replaced with ENABLE

SQL Command

Data Definition

Operation	Details
DROP	**DROP TABLE** <table name>; **ALTER TABLE** <table name> **DROP** <column name>; **ALTER TABLE** <table name> **SET UNUSED** <column list>; **ALTER TABLE** <table name> **DROP UNUSED COLUMNS;**
ALTER Structure of Table	**ALTER TABLE** < table name> **MODIFY** <column name DATATYPE(size);
RENAME	**RENAME** <table name1> **TO** <table name2> ; **ALTER TABLE** < table name> **RENAME** <column1> **TO** <column2> ;
TRUNCATE	**TRUNCATE TABLE** < table name> ; **DELETE FROM** <table name> **WHERE** <condition> **COMMIT;**

ALTER TABLE <table_name> ADD <COL_NAME> <COL_TYPE>

TRUNCATE TABLE <table_name> => P.S: this command remove the data physically so no rollback is possible, but the delete remove them logically so we can do roll back

ALTER TABLE <table_name> set unused (col_list) => this command is used to mark that some column are not used in our table so our select statement cannot reach them

Alter Table <table_name> DROP UNUSED Columns => is to drop all the unused columns

Insert, Update, Delete, Merge(Insert values from table to the other)

Operation	Details
INSERT	**INSERT INTO** <table name> (<column list>) **VALUES** (data list)
UDATE	**UPDATE** <table name> SET column1= <value1> , column2=<value2> **WHERE** <condition>
DELETE	**DELETE FROM** <table name> **WHERE** <condition>
MERGE	**MERGE INTO** <table> t **USING** <data set> **ON** (<condition>) **WHEN MATHCHED THEN** **UPDATE SET** col1 =<> , col2 = <> **WHEN NOT MATHCHED THEN** **INSERT** (t.col1,t.col2) **VALUES** (val1,val2)

INSERT INTO <table_name> (col1,col2,col3....) values (val1,val2,val3....) => P.S : we insert all cols excepte for the auto-generated and virtual columns

```
MERGE INTO TargetTable
USING SourceTable
ON Condition
WHEN MATCHED THEN
UPDATE SET col_1 = value_1, col_2 =
value_2...col_n = value_n
```

```
WHERE <UpdateCondition>
[DELETE WHERE <DeleteCondition>] WHEN NOT MATCHED
THEN
INSERT (col_1,col_2...col_n)
Values(value_1,value_2...value_n)
WHERE <InsertCondition>;
```

```
MERGE INTO emp e1
USING emp_test e2 ON (e1.empno = e2.empno)
WHEN MATCHED THEN
UPDATE SET e1.sal = e2.sal
WHEN NOT MATCHED THEN
INSERT (e1.EMPNO, e1.ENAME,e1.JOB,e1.MGR,e1.HIREDATE,e1.SAL,e1.COMM,e1.DEPTNO)
VALUES (e2.EMPNO, e2.ENAME,e2.JOB,e2.MGR,e2.HIREDATE,e2.SAL,e2.COMM,e2.DEPTNO)
```

We are inserting/updating the e1 table from e2 on a condition specified by the ON statement

If match then Update if not than insert

Multiple Insert

Insert ALL
Into <table_name> (col1,colN) Values (value1....,valueN)
Into <table_name> (col1,colN) Values (value1....,valueN)
Into <table_name_2> (col1,colN) Values (value1....,valueN)
Select * from DUAL;

Grant and Revoke

Grant <operation like Select,Update, delete, Drop ..> ON <resource_name like user1.table1> TO user2

REVOKE <operation like Select,Update, delete, Drop ..> ON <resource_name like user1.table1> TO user2

P.S: If we log as user_1 by default the grant .... ON Resource will be the Resource of user_1 so it equivalent to Grant ... user_1.resource

So the second user need to mention user_1.resource to get this grant to work

Transactions (Commit,Rollback and SavePoint)

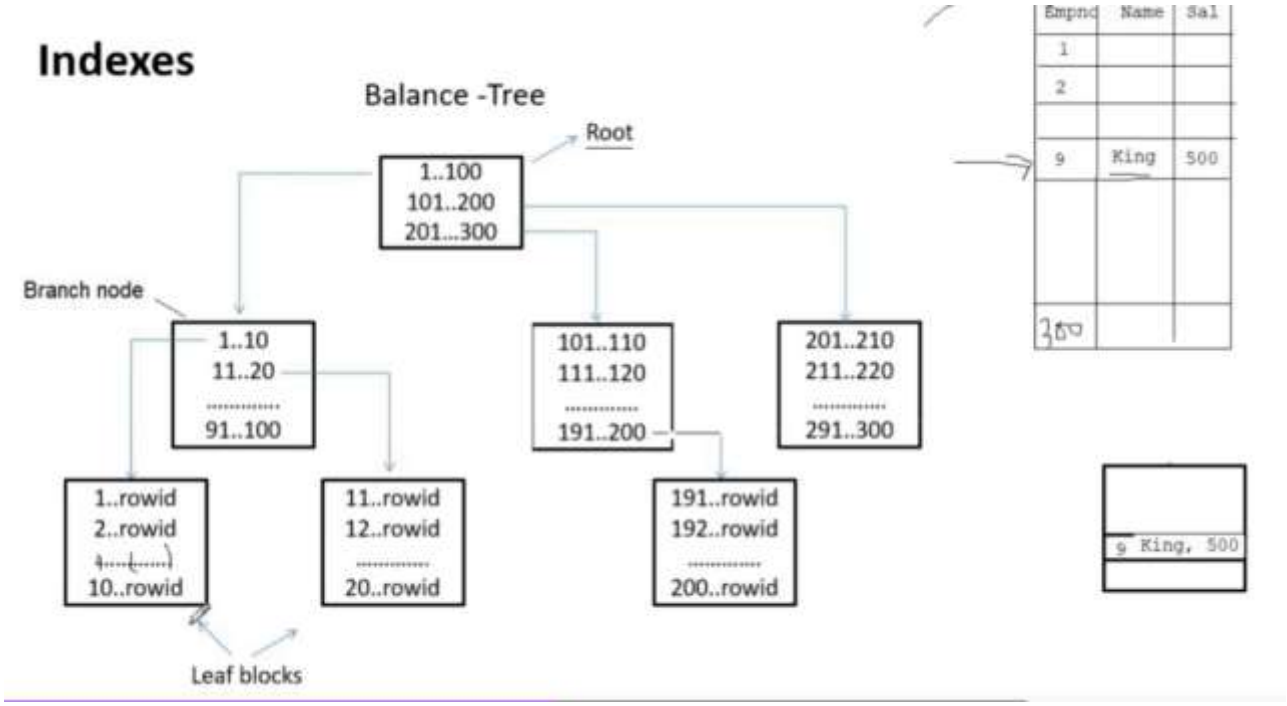
Command	Description
COMMIT	Commit command permanently saves any transaction into database
ROLLBACK	This command restores the database to last committed state.
TO SAVE-POINT	SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required

Save Point means simply like a checkpoint, for example we can rollback the entire process or we can roll back to a specific point (we can rollback data to the middle of the transaction execution).

To use Save Point we can simply : SAVEPOINT <save_point_name>; Rollback to <save_point_name>

Indexes

B-Tree



B-Tree basically is like a binary tree with multiple level, the last branch or the leaf is actual data. The B-Tree use the same logic of the binary tree. The leaf or the last branch contains the actual data.

Creation of B-Tree Index

Create Index <index_name> on <table_name> (cols_list...)

Bitmap Index

Create BitMap Index <index_name> on <table_name> (cols_list...)

Function Index

Create Index <index_name> on <table_name> (FUNCTION(COL1,COL2....))

A function-based index reduces computation for the database. If you have a query that consists of expression and use this query many times, the database has to calculate the expression each time you execute the query. To avoid these computations, you can create a function-based index that has the exact expression.

Flash Back (Restore What's deleted)

In oracle when we do a delete or drop query without the purge we are only deleting it logicaly

Flashback Table

SET recyclebin = ON; =>we tell oracle that when we delete/drop we only move to a recycle bin if OFF then it will be a permanent delete

SELECT * from user_recyclebin => user_recyclebin is table that store info about the deleted objects. From this query we have a column called object_name that is table so simply we can query it by : **select * from "<object_name>"**

FLASHBACK TABLE <DELETED_TABLE_NAME> TO BEFORE DROP => this query is used to restore the table after drop

Permanent Delete

DROP TABLE <table_name> PURGE => here we permanently deleted a table so there is no way to get it back

PURGE RECYCLEBIN => this simply to permanently remove object and their data from the recycle bin and from the disk

By purge we mean we will remove the data from dish so more store storage

Flashback query

Flash back query is used to retrieve historical data. Data that have been removed or overwritten.

The below query is used to retrieve historical data until a certain offset
SELECT * FROM <table_name> AS OF TIMESTAMP SYSDATE - <number_of_day_in history>

The below query is to set for how many minutes oracle will hold history data:
ALTER SYSTEM SET DB_FLASHBACK_RETENTION_TARGET = <number of minutes>

Sequence and Synonyms

Sequence

```
CREATE SEQUENCE schema_name.sequence_name
[INCREMENT BY interval]
[START WITH first_number]
[MAXVALUE max_value | NOMAXVALUE]
[MINVALUE min_value | NOMINVALUE]
[CYCLE | NOCYCLE]
[CACHE cache_size | NOCACHE]
[ORDER | NOORDER];
```

```
SELECT seq.NEXTVAL , seq.CURRVAL FROM dual
DROP SEQUENCE <sequence name>
```

Keyword	Description
START WITH	Starting value of the sequence
MAXVALUE	Maximum value of the sequence
NOMAXVALUE	NOMAXVALUE to denote a maximum value of 10^27
MINVALUE	Minimum value of the sequence
NOMINVALUE	NOMINVALUE indicate a minimum value of 1
CYCLE	CYCLE to allow the sequence to generate value after it reaches the limit
NOCYCLE	Sequence to stop generating after reaching max value
CACHE	Specify the number of sequence values that will pre allocate in the memory for faster access

ORDER clause in sequence is only meaningful in **RAC**.

It guarntees the generation of sequence in order, no matter which instance received the request.

If you dont use ORDER then to illustrate, assume a sequence defined with cache=20. Instance 1 has sequence values 1 through 20 in its cache. Instance 2 has sequence values 21 through 40 in its cache. Normally, concurrent sessions might generate sequence values in this order: 1, 2, 21, 3, 22, 4, 23, and 24.but with ORDER clause this values will be 1,2,3,4,5,6,7,..

Hence, It is mentioned in the document that if the purpose of the sequence is to generate unique values then ORDER is not needed but it is needed if sequence is used to define chronological order in RAC.

Cache: If you specify the Cache in sequence as 20 then oracle takes 20 value in a bunch and put its value in SGA and data dictionary is updated once. So if you want to use 35 sequence values then there will be only 2 times when data dictionary is updated improving the performance against 35 updates in the data dictionary in case of NO CACHE. The cache is used to improve the performance of the sequence. But also in database shutdown, you will lose unused buffered sequence values.

Synonyms

What is a synonym

A synonym is an alias for a schema object. It is use full when we need to mask the name of the owner and the table name

We have to types of synonyms : Public and Private.

Public is to hide the schema and the object name: owner.object1 => synonyme

Private is to hide only the object name: user.object1 => user1. synonyme

How to create

Private Synonym => CREATE SYNONYM <synonym_name> FOR <table_name>

Public Synonym => CREATE PUBLIC SYNONYM <synonym_name> FOR <table_name>

Managing Data Dictionary

Definition

Data Dictionary are basically readonly views. The name of those views is usually in the format of : <PREFIX>_OBJECT_NAME

Example: USER_TABLES or ALL_TABLES

prefix	Scope
USER	User view (What is in user's schema)
ALL	Expanded user's view (What user can access)
DBA	Database administrator's view (what is in all users' schemas)

Some Views

User_tables => the tables created by the user with the tablespace and many more info

User_segments => contains information about table size and partitions

USER_TAB_COLS => to get info about the exiting columns for each user table

The same views can be applied with the ALL prefix

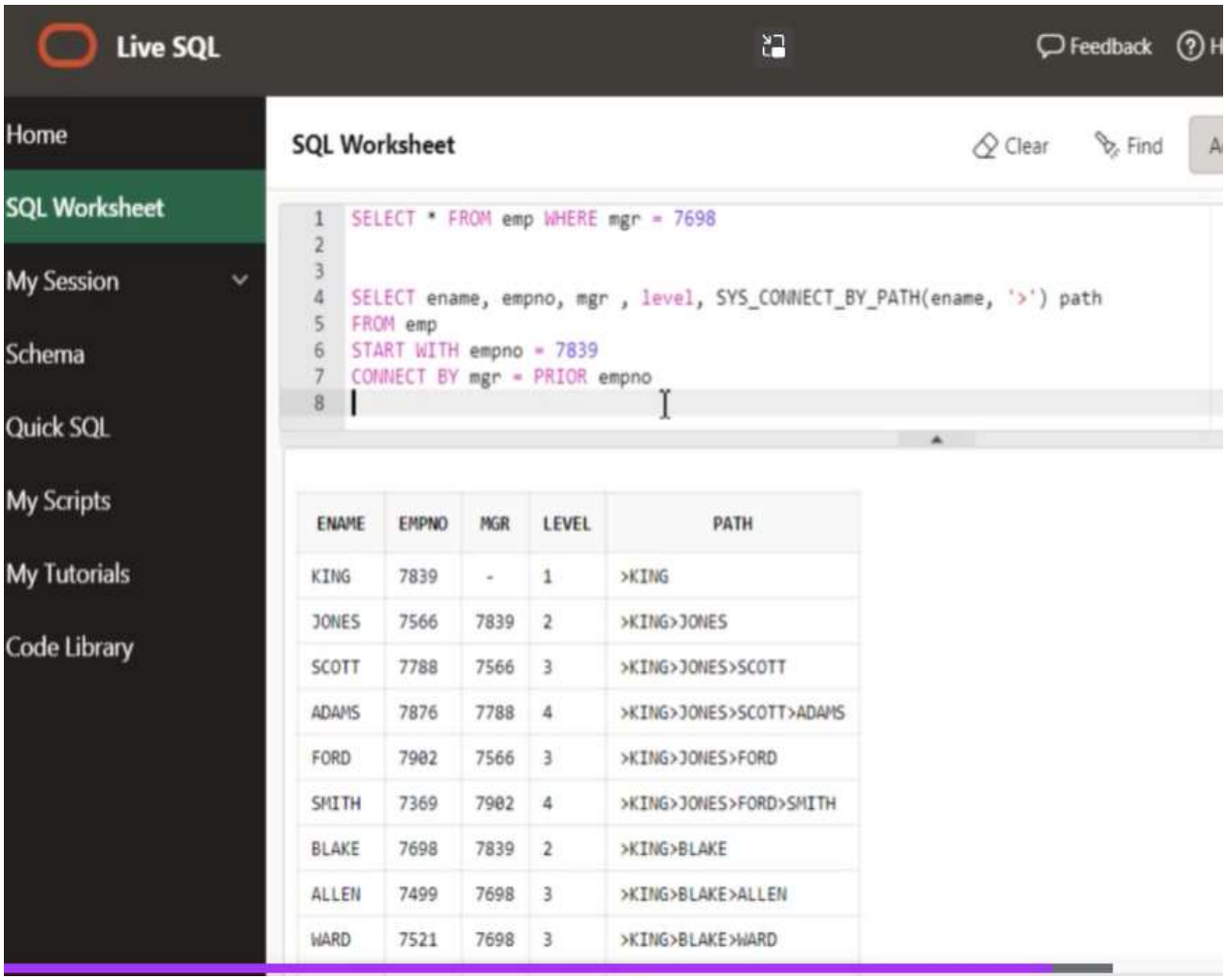
USER/ALL/DBA	Description
<Prefix>_TABLE	Find the Tables
<Prefix>_OBJECTS	Find Oracle objects
<Prefix>_SEQUENCE	Find the Sequence
<Prefix>_VIEWS	Find the views
<Prefix>_SYNONYMS	Find the synonyms
<Prefix>_CONSTRAINTS	Find the Constraints
<Prefix>_TAB_COLS	Find the columns of Table
DBA_ROLES	Find all the Roles
<Prefix>_USERS	Find all users
<Prefix>_TAB_PRIVS	Find all table Privileges of a user
<Prefix>_ROLE_PRIVS	Find all Privileges attached to a role

Hierarchal Data (Graphing Node/vertex)

Functions

The Prior mean match the Non Prior column of the current Row with a previous row at the prior column. Means we area matching the mgr of the current row with empno of the previous row

CONNECT BY PRIOR	[**START WITH** condition] **CONNECT BY** [**NOCYCLE**] condition
LEVEL	Sudo column
CONNECT BY NOCYCLE/CYCLE(Default)	[**START WITH** condition] **CONNECT BY** [**NOCYCLE**] condition
SYS_CONNECT_BY_PATH	**SYS_CONNECT_BY_PATH**(column, char)
CONNECT_BY_ROOT	It displays the root value
CONNECT_BY_ISLEAF	If show 1 or 0 1 = leaf node, 0 = branch node
CONNECT_BY_ISCYCLE	If show 1 or 0 1= It is cyclic , 0 = not cyclic



Privileges

System Privileges

System Privileges	Object Privileges
GRANT CREATE TABLE TO <user>	**GRANT SELECT** ON <table> TO <user>
GRANT CREATE VIEW TO <user>	**GRANT SELECT ON** <view name> TO <user>
GRANT CREATE DIRETORY TO <user>	**GRANT READ ON** <directory name> TO <user>
GRANT CREATE SEQUENCE TO <user>	**GRANT SELECT ON** <sequence name> TO <user>

GRANT CREATE ANY TABLE TO … => means the user can create tables in all the schemas he wants

All the privileges of a Table – (1ZO-071)
CREATE TABLE/CREATE ANY TABLE - System privileges
SELECT, INSERT , UPDATE, DELETE ON <Table> --Object Privileges
ALL ON <Table> --Object Privileges

USER_SYS_PRIVS => table that have system privileges of the current user

USER_TAB_PRIVS => table that have the object privileges of the current user

Role

GRANT <role_name> TO <user_name> => to assign a user with a role

CREATE ROLE <ROLE_NAME> => to create the role

GRANT <INSERT,UPDATE ….> ON schema.table TO <ROLE_NAME> => to set the role's right

USER_TAB_PRIVS => can be used to see the privileges assigned to a role, DBA_ROLES also is a table used to find all the created roles

Custom Aggregation Function

```
create /*or replace*/ type body mult_agg_type is

   static function ODCIAggregateInitialize(sctx IN OUT mult_agg_type) return number is
   begin
      sctx := mult_agg_type (null);
      return ODCIConst.Success;
   end;
   member function ODCIAggregateIterate(self IN OUT mult_agg_type, value IN number)
return number is
   begin
      if self.totalAggValue is null then
         self.totalAggValue := value;
      else
         self.totalAggValue := self.totalAggValue*value;
      end if;
      return ODCIConst.Success;
   end;
   member function ODCIAggregateTerminate(self IN mult_agg_type, returnValue OUT
number, flags IN number) return number is
   begin
      returnValue := self.totalAggValue;
      return ODCIConst.Success;
   end;
   member function ODCIAggregateMerge(self IN OUT mult_agg_type, ctx2 IN
mult_agg_type) return number is
   begin
      self.totalAggValue := self.totalAggValue*ctx2.totalAggValue;
      return ODCIConst.Success;
   end;
end;


create /*or replace*/ function agg_product(input number) RETURN number
   PARALLEL_ENABLE AGGREGATE using mult_agg_type;
```

Select agg_product(col) from.... Groupby

Partitioning Tables and Indexes

KEYWORDS

Oracle Reserved Words

ACCESS	ELSE	MODIFY	START
ADD	EXCLUSIVE	NOAUDIT	SELECT
ALL	EXISTS	NOCOMPRESS	SESSION
ALTER	FILE	NOT	SET
AND	FLOAT	NOTFOUND	SHARE
ANY	FOR	NOWAIT	SIZE
ARRAYLEN	FROM	NULL	SMALLINT
AS	GRANT	NUMBER	SQLBUF
ASC	GROUP	OF	SUCCESSFUL
AUDIT	HAVING	OFFLINE	SYNONYM
BETWEEN	IDENTIFIED	ON	SYSDATE
BY	IMMEDIATE	ONLINE	TABLE
CHAR	IN	OPTION	THEN
CHECK	INCREMENT	OR	TO
CLUSTER	INDEX	ORDER	TRIGGER
COLUMN	INITIAL	PCTFREE	UID
COMMENT	INSERT	PRIOR	UNION
COMPRESS	INTEGER	PRIVILEGES	UNIQUE
CONNECT	INTERSECT	PUBLIC	UPDATE
CREATE	INTO	RAW	USER
CURRENT	IS	RENAME	VALIDATE
DATE	LEVEL	RESOURCE	VALUES
DECIMAL	LIKE	REVOKE	VARCHAR
DEFAULT	LOCK	ROW	VARCHAR2
DELETE	LONG	ROWID	VIEW
DESC	MAXEXTENTS	ROWLABEL	WHENEVER
DISTINCT	MINUS	ROWNUM	WHERE
DROP	MODE	ROWS	WITH

Oracle Keywords

ADMIN	CURSOR	FOUND	MOUNT
AFTER	CYCLE	FUNCTION	NEXT
ALLOCATE	DATABASE	GO	NEW
ANALYZE	DATAFILE	GOTO	NOARCHIVELOG
ARCHIVE	DBA	GROUPS	NOCACHE

ARCHIVELOG	DEC	INCLUDING	NOCYCLE
AUTHORIZATION	DECLARE	INDICATOR	NOMAXVALUE
AVG	DISABLE	INITRANS	NOMINVALUE
BACKUP	DISMOUNT	INSTANCE	NONE
BEGIN	DOUBLE	INT	NOORDER
BECOME	DUMP	KEY	NORESETLOGS
BEFORE	EACH	LANGUAGE	NORMAL
BLOCK	ENABLE	LAYER	NOSORT
BODY	END	LINK	NUMERIC
CACHE	ESCAPE	LISTS	OFF
CANCEL	EVENTS	LOGFILE	OLD
CASCADE	EXCEPT	MANAGE	ONLY
CHANGE	EXCEPTIONS	MANUAL	OPEN
CHARACTER	EXEC	MAX	OPTIMAL
CHECKPOINT	EXPLAIN	MAXDATAFILES	OWN
CLOSE	EXECUTE	MAXINSTANCES	PACKAGE
COBOL	EXTENT	MAXLOGFILES	PARALLEL
COMMIT	EXTERNALLY	MAXLOGHISTORY	PCTINCREASE
COMPILE	FETCH	MAXLOGMEMBERS	PCTUSED
CONSTRAINT	FLUSH	MAXTRANS	PLAN
CONSTRAINTS	FREELIST	MAXVALUE	PLI
CONTENTS	FREELISTS	MIN	PRECISION
CONTINUE	FORCE	MINEXTENTS	PRIMARY
CONTROLFILE	FOREIGN	MINVALUE	PRIVATE
COUNT	FORTRAN	MODULE	PROCEDURE

PROFILE	SAVEPOINT	SQLSTATE	TRACING
QUOTA	SCHEMA	STATEMENT_ID	TRANSACTION
READ	SCN	STATISTICS	TRIGGERS
REAL	SECTION	STOP	TRUNCATE
RECOVER	SEGMENT	STORAGE	UNDER
REFERENCES	SEQUENCE	SUM	UNLIMITED
REFERENCING	SHARED	SWITCH	UNTIL
RESETLOGS	SNAPSHOT	SYSTEM	USE
RESTRICTED	SOME	TABLES	USING
REUSE	SORT	TABLESPACE	WHEN

ROLE	SQL	TEMPORARY	WRITE
ROLES	SQLCODE	THREAD	WORK
ROLLBACK	SQLERROR	TIME	

PL/SQL Reserved Words

ABORT	BETWEEN	CRASH	DIGITS
ACCEPT	BINARY_INTEGER	CREATE	DISPOSE
ACCESS	BODY	CURRENT	DISTINCT
ADD	BOOLEAN	CURRVAL	DO
ALL	BY	CURSOR	DROP
ALTER	CASE	DATABASE	ELSE
AND	CHAR	DATA_BASE	ELSIF
ANY	CHAR_BASE	DATE	END
ARRAY	CHECK	DBA	ENTRY
ARRAYLEN	CLOSE	DEBUGOFF	EXCEPTION
AS	CLUSTER	DEBUGON	EXCEPTION_INIT
ASC	CLUSTERS	DECLARE	EXISTS
ASSERT	COLAUTH	DECIMAL	EXIT
ASSIGN	COLUMNS	DEFAULT	FALSE
AT	COMMIT	DEFINITION	FETCH
AUTHORIZATION	COMPRESS	DELAY	FLOAT
AVG	CONNECT	DELETE	FOR
BASE_TABLE	CONSTANT	DELTA	FORM
BEGIN	COUNT	DESC	FROM

PL/SQL Reserved Words (continued):

FUNCTION	NEW	RELEASE	SUM
GENERIC	NEXTVAL	REMR	TABAUTH
GOTO	NOCOMPRESS	RENAME	TABLE
GRANT	NOT	RESOURCE	TABLES
GROUP	NULL	RETURN	TASK
HAVING	NUMBER	REVERSE	TERMINATE
IDENTIFIED	NUMBER_BASE	REVOKE	THEN
IF	OF	ROLLBACK	TO
IN	ON	ROWID	TRUE
INDEX	OPEN	ROWLABEL	TYPE
INDEXES	OPTION	ROWNUM	UNION

INDICATOR	OR	ROWTYPE	UNIQUE
INSERT	ORDER	RUN	UPDATE
INTEGER	OTHERS	SAVEPOINT	USE
INTERSECT	OUT	SCHEMA	VALUES
INTO	PACKAGE	SELECT	VARCHAR
IS	PARTITION	SEPARATE	VARCHAR2
LEVEL	PCTFREE	SET	VARIANCE
LIKE	POSITIVE	SIZE	VIEW
LIMITED	PRAGMA	SMALLINT	VIEWS
LOOP	PRIOR	SPACE	WHEN
MAX	PRIVATE	SQL	WHERE
MIN	PROCEDURE	SQLCODE	WHILE
MINUS	PUBLIC	SQLERRM	WITH
MLSLABEL	RAISE	START	WORK
MOD	RANGE	STATEMENT	XOR
MODE	REAL	STDDEV	
NATURAL	RECORD	SUBTYPE	

Hints

1. We can use Cross APPLY to reuse precalculated values or we can use the WITH statement
2. In case we need to enrich table 1 with details from table2 its better to use joins and not filtered cartesian join for performance optimization
3. For security
    a. we can use View for limiting access for data
    b. using Synonyms to hide schema owner and object names
4. To drop multiple columns we should use the unused alter statement and then a alter .. drop unused columns
5. Some time we can use oracle hints to optimize the execution of oracle sql