Advance SQL

Part-2

VIEW WITH READ ONLY CONSTRAINT

The WITH READ ONLY option allows the user to create a read-only view. You cannot use the DELETE, INSERT or UPDATE commands to modify data for the view.

To create a read only view “prodreadonly” for all products in the product table which have a unit_price more than 100.

```
CREATE or REPLACE VIEW prodreadonly AS

SELECT * FROM product WHERE unit_price > 100

WITH READ ONLY CONSTRAINT view_no;
```

Dropping VIEWS

To remove views we use the command DROP VIEW. You can change the definition of a view by dropping and re-creating it. If you delete a view, you can no more access the virtual tables based on the view.

Syntax:

```
DROP VIEW <view name>
```

3.4.2 INDEXES

Index is an object which can be defined as the ordered list of values of a column or combination of columns used for faster searching and sorting of data. An index speeds up joins and searches by providing a way for a database management system to go directly to a row rather than having to search through all the rows until it finds the one you want. By default, indexes are created for primary keys and unique constraints of a table. However, creating indexes must be avoided on columns that are updated frequently since this slows down insert, update and delete operations.

Creating Indexes

To create an index, you need to use the CREATE INDEX command. In addition, you can use the UNIQUE keyword to specify that an index contains only unique values.

Syntax:

```
CREATE [UNIQUE] INDEX index_name
ON table_name (column_name1 [ASC | DESC] [, column_name2 [ASC | DESC]]......)
```

Removing Indexes

To remove or delete an index, we need to use the DROP INDEX command.

Syntax:

```
DROP INDEX <index name>
```

3.4.3 SEQUENCES

SEQUENCE is a type of database object which can be used to generate numbers in a sequence. This can be used to generate values for primary keys.

Create Sequence

Sequences can be created using the CREATE SEQUENCE command which has the following syntax:

```
CREATE SEQUENCE <sequence name>
START WITH <integer-value>
INCREMENT BY <integer-value>
MAXVALUE <integer-value> OR NOMAXVALUE
MINVALUE <integer-value> OR NOMINVALUE
CYCLE OR NOCYCLE
CACHE OR NOCACHE
ORDER OR NOORDER
```

START WITH <integer-value>: specifies the 1st sequence number to be generated.

INCREMENT BY <integer-value>: The integer number by which sequence number should be incremented for generating the next number. If it is positive then values are ascending and if it is negative then values are descending. The default value is 1.

MAXVALUE <integer-value>: If the increment value is positive then MAXVALUE determines the maximum value up to which the sequence numbers will be generated.

NOMAXVALUE: Specifies the maximum value of 10^27 for an ascending sequence or -1 for a descending sequence.

MINVALUE <integer-value>: If the increment value is negative then MINVALUE determines the minimum value up to which the sequence numbers will be generated.

NOMINVALUE: Specifies the minimum value of 1 for an ascending sequence or -10^26 for a descending sequence.

CYCLE: Causes the sequences to automatically recycle to minvalue when maxvalue is reached for ascending sequences; for descending sequences, it causes to recycle from minvalue back to maxvalue.

NOCYCLE: Sequence numbers will not be generated after reaching the maximum value for ascending sequences or minimum value for descending sequences.

CACHE: Specifies how many values are pre-allocated in buffers for faster access. Default value is 20.

NOCACHE: Sequence numbers are not pre-allocated.

ORDER: Generates the number in a serial order.

NOORDER: Generates the number in a random order.

Initializing and Accessing Sequence

A sequence needs to be initialized before being used. Every sequence is initialized by a pseudo column NEXTVAL. Once you've created a sequence, you typically use it within an INSERT statement. The NEXTVAL pseudo column gets the next value from the sequence so it can be inserted into the table. The CURRVAL pseudo column is used to check the current value of the sequence.

Modifying Sequence

It may be required later on to change certain parameters of an already created sequence. This can be done using the ALTER SEQUENCE command.

Syntax:

ALTER SEQUENCE <sequence name>

[sequence attributes]

Dropping Sequence

A sequence can be deleted using the DROP SEQUENCE command.

Syntax:

DROP SEQUENCE <sequence name>

4. SYNONYMS

Synonyms are alternative names for an existing object which are permanently stored in the database. We have used alias names for accessing tables with shorter names in various queries. The difference is that these alias names are of temporary nature and are lost from one query to another, whereas synonyms are permanent alias names for objects.

Advantages of using synonyms

- Synonyms are often used for security and convenience.
- They can do the following things:
  - They can hide or mask the name and owner of an object.
  - Provide location transparency for remote objects of a distributed database.
  - Simplify SQL statements for database users.
- With the help of synonyms the user can insert, update or retreive records from any database object. Synonyms cannot be used in a DROP TABLE, DROP VIEW or TRUNCATE TABLE statement.

The various objects for which synonyms can be created are as follows:

- Tables
- Views
- Materialized Views
- Stored Function
- Stored Procedures
- Packages
- Sequences

- Synonyms

There are two kinds of synonyms – public and private.

Public Synonym: These are accessible to all users provided they have the appropriate object privilege on the object on which the synonym is created.

Private Synonym: These belong only to the user who creates it.

Creating Synonyms

Synonyms can be created using the CREATE SYNONYM command

Syntax:

CREATE [PUBLIC] SYNONYM <synonym name> FOR <object name>

Renaming Synonyms

Only Private Synonyms can be renamed using the Rename statement.

Syntax:

RENAME <old synonym name> TO <new synonym name>

Modifying a Synonym

To modify or alter or change a synonym use the OR REPLACE clause. You can use this clause to change the definition of an existing synonym without dropping it.

Syntax:

CREATE OR REPLACE [PUBLIC] SYNONYM <synonym name> FOR <object name>

Removing a Synonym

Synonyms can be removed or deleted using the DROP SYNONYM statement.

Syntax:

DROP SYNONYM <synonym name>

3.5 SUBQUERIES

Queries can be nested so that the results of one query can be used in another query via a relational operator or aggregation function. A nested query is known as a **subquery**. While joins and other table operations provide computationally superior (i.e. faster) alternatives in many cases, the use of subqueries introduces a hierarchy in execution which can be useful or necessary.

Since you know how to code SELECT statements, you already know how to code a subquery. It's simply a SELECT statement that's coded within another SQL statement. A subquery can return a single value, a result set that contains a single column (single row subquery), or a result set that contains one or more columns (multiple row subquery).

The following is the list of comparison operators used in a single row subquery.

Symbol	**Description**
	Equal
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
<>	Not equal
!=	Not equal

The following is the list of comparison operators used in multiple row subquery.

Subquery Operator	**Description**
IN, =ANY	Look for a match in any of the subquery rows
>ANY	Look for a value greater than any of the subquery rows

<ANY	Look for a value less than any of the subquery rows
>ALL	Look for a value greater than all of the subquery rows
<ALL	Look for a value less than all of the subquery rows
!=ALL, NOT IN	Look for a value not present in any of the subquery rows

Four ways to introduce a subquery in a SELECT statement

- In a WHERE clause as a search condition.
- In a HAVING clause as a search condition.
- In a FROM clause as a table specification.
- In a SELECT clause as a column specification.

Examples for subqueries:

- Get the product name and company name of the products which have the maximum price.

```
SELECT product_name, company_name FROM product
WHERE unit_price = (SELECT MAX(unit_price) FROM product);
```

- Get the names of products which have a unit price less than or equal to the average price of the products.

```
SELECT product_name FROM product
WHERE unit_price <= (SELECT AVG(unit_price) FROM product);
```

- Get the names of the product which has the highest price.

```
SELECT product_name FROM product
WHERE unit_price >= ALL (SELECT unit_price FROM product);
```

- Get the names of the products which have been ordered.

```
SELECT product_name FROM product
WHERE product_id IN (SELECT product_id FROM order_prod);
```

Or

SELECT product_name FROM product

WHERE product_id = ANY (SELECT product_id FROM order_prod);

- Get the names of the products which have not been ordered by any customers.

SELECT product_name FROM product

WHERE product_id NOT IN (SELECT product_id FROM order_prod);

Or

SELECT product_name FROM product

WHERE product_id != ALL (SELECT product_id FROM order_prod);

- Get the names of the products which have been ordered in maximum quantity.

SELECT product_name FROM product

WHERE product_id IN (SELECT product_id FROM order_prod

GROUP BY product_id HAVING SUM(total_units) > = ALL

(SELECT SUM(total_units) FROM order_prod GROUP BY product_id));

3.5.1 SUBQUERY in DDL and DML commands

Subqueries can be used in DDL commands to create a new table from an existing table. The subquery is used to retreive the data using which the new table is created. The structure of the new table will be same as the structure of the query.

Subquery can be used to insert, update and delete rows from the existing table.

For example,

- To create a table student_new containing roll, name and grade of students enrolled in semester 3.

CREATE TABLE student_new AS

(SELECT roll, name, grade FROM student WHERE semester = 3);

- To insert a record in the table student_new (created above) with roll number 1 more than the maximum roll number of the table.

```
INSERT INTO student_new VALUES

((SELECT MAX(roll) + 1 FROM student_new), ’Pravin’, ’A’);
```

6. SUMMARY

- Aggregate functions operate on a series of values and return a single summary value.

- Aggregate functions are commonly used with the GROUP BY clause in a SELECT statement, where the rows of a queried table are divided into groups.

- The most common aggregate functions are
  - AVG
  - SUM
  - MIN
  - MAX
  - COUNT
- GROUP BY clause forms groups on the specified columns. The GROUP BY clause is used alongwith the aggregate functions to retreive data grouped according to one or more columns.
- The HAVING clause to restrict the groups of returned rows to those groups for which the specified condition is TRUE.
- A view is a logical representation of one or more tables. In essence, a view is a stored query.

- Views can be classified as updateable views and non-updateable views.
- Index is an object which can be defined as the ordered list of values of a column or combination of columns used for faster searching and sorting of data.
- By default, indexes are created for primary keys and unique constraints of a table.
- Creating indexes must be avoided on columns that are updated frequently since this slows down insert, update and delete operations.
- SEQUENCE is a type of database object which can be used to

generate numbers in a sequence. This can be used to generate values for primary keys.

- Synonyms are alternative names for an existing object which are permanently stored in the database.
- There are two kinds of synonyms – public and private.
- Queries can be nested so that the results of one query can be used in another query via a relational operator or aggregation function. A nested query is also known as a subquery.
- Subqueries can be used in DDL commands to create a new table from an existing table.
- Subquery can be used to insert, update and delete rows from the existing table.

7. REVIEW QUESTIONS

1. What are aggregate functions? explain in detail.
2. Explain the Group By Having clause with examples.
3. Differentiate between the WHERE clause and HAVING clause?
4. What is view? What are the benefits of using views?
5. Explain the types of views in detail.
6. Write a short note on indexes.
7. What is a sequence? Explain the syntax for creating a sequence.
8. What is a synonym? Why should you use a synonym?
9. Explain the different types of synonyms? List the objects for which synonyms can be created.
10. What is a subquery? Explain in detail with examples.

8. LAB ASSIGNMENT

1. Create a CUSTOMER table with the following columns and constraints

Column name	Data type	Size	Constraint
CUSTOMER_ID	CHAR	6	PRIMARY KEY. MUST BEGIN WITH 'C'
CUSTOMER_NAME	VARCHAR2	20	NOT NULL

ADDRESS	VARCHAR2	20	UNIQUE
CITY	VARCHAR2	20	
PINCODE	NUMBER	6	
STATE	VARCHAR2	20	
BALANCE_DUE	NUMBER	8,2	

2. Create a PRODUCT table with the following columns and constraints

Column name	Data type	Size	Constraint
PRODUCT_CODE	CHAR	6	PRIMARY KEY
PRODUCT_NAME	VARCHAR2		UNIQUE
QTY_AVAIL	NUMBER	5	
COST_PRICE	NUMBER	8,2	
SELLING_PRICE	NUMBER	8,2	

3. Create a ORDER table with the following columns and constraints

Column name	Data type	Size	Constraint
ORDER_NO	CHAR	6	
ORDER_DATE	TIMESTAMP		
CUSTOMER_ID	CHAR	6	
PRODUCT_CODE	CHAR	6	
QUANTITY	NUMBER	5	

PRIMARY KEY = ORDER_NO + ORDER_DATE + CUSTOMER_ID + PRODUCT_CODE

4. Insert 5-10 records in all the tables.
5. Apply UNIQUE constraint on CUSTOMER_ID+PRODUCT_CODE on the ORDER table.
6. Define a foreign key on CUSTOMER_ID of ORDER table referring to CUSTOMER_ID of CUSTOMER table.
7. Define a foreign key on PRODUCT_CODE of ORDER table referring to PRODUCT_CODE of PRODUCT table.
8. Count the customerwise number of orders.
9. Calculate the average selling price of all products.
10. List the customer names for which we have orders in hand.
11. List the yearwise number of orders placed.
12. Display the customers who have placed some order. Use IN and EXISTS operator.
13. Find the customer who has placed maximum number of orders.
14. Create unique index on ORDER_NO, CUSTOMER_ID and PRODUCT_CD of ORDER table.
15. Create a non unique index on STATE of CUSTOMER table.
16. Create a view named "vw_balance" which will display customer names with balances more than 4000.
17. Add two new customers through the view.
18. Create a view named "vw_city" which will contain citywise total balances.

3.9 BIBLIOGRAPY, REFERENCES AND FURTHER READING

- Database Management Systems, Third Edition by RamaKrishnan, Gehre. McGraw Hill
- Database System Concepts, Fifth Edition by Silberschatz, Korth, Sudarshan. McGraw Hill
- Murach's Oracle SQL and PL/SQL by Joel Murach. Shroff Publishers & Distributors
- Oracle Database 11g by Satish Asnani. PHI Learning Private Limited
- Oracle 11g: PL/SQL Reference Oracle Press.
- Expert Oracle PL/SQL, By: Ron Hardman,Michael McLaughlin, Tata McGraw-Hill

- SQL, PL/SQL The programming language of Oracle, Bayross Ivan, BPB Publications
- Fundamentals of Database Systems, Elmasri Ramez and Navathe B. Shamkant, Pearson

3.10 ONLINE REFERENCES

Wikipedia Link

http://en.wikipedia.org/wiki/SQL

Oracle Database PL/SQL language Reference 11g Release 2 (11.2), part number E25519-05

http://docs.oracle.com/cd/E11882_01/appdev.920/a96590/adg09dyn.htm

Unit - II

4

SECURITY PRIVILEGES, SET OPERATORS & DATETIME FUNCTIONS

Unit Structure

1. OBJECTIVES

At the end of this chapter you will be able to:

- Get summary information using CUBE, ROLLUP and GROUPING
- Combine queries using SET operators
- Use DateTime Functions
- Control User Access

2. INTRODUCTION

In the previous unit, we saw basic SQL operators, statements, queries and subqueries. In this unit, we shall see advanced subqueries, various advanced operators and security privileges.

In the previous chapter we discussed SQL keywords and functions. In this chapter we will see the extensions of GROUP BY clause: ROLLUP and CUBE operators, and GROUPING function. Also we will combine queries using SET operators and see various DateTime functions.

3. ENHANCEMENTS TO GROUP BY FUNCTION

In the previous chapter we saw that the **GROUP BY** clause forms groups on specified columns. The **HAVING** clause filters these groups depending on some conditions.

Assume the records in the EMP table as shown below.

EMPNO	ENAME	HIREDATE	DEPTNO	JOB	SALARY	COMM
111	Satish	19-DEC-2008	10	CLERK	7000	1000
222	Rashmi	01-JAN-1987	20	ANALYST	8000	550
333	Rishi	05-JUN-1976	10	MANAGER	10,000	450
444	Anil	16-APR-1967	10	PRESIDENT	15,000	2000
555	Anita	-	30	MANAGER	8000	1000
666	Nilesh	20-MAY-1987	20	MANAGER	13,000	-
777	Ruchi	11-JUN-2000	30	SALESMAN	5000	-
888	Sarika	-	20	SALESMAN	4000	-

A simple GROUP BY clause will show the following result.

- Display the amount of salary being paid jobwise for each department.

```
SELECT deptno, job, SUM(sal) FROM emp

GROUP BY deptno, job

ORDER BY deptno, job;
```

DEPTNO	JOB	SUM(SAL)
10	CLERK	7000
10	MANAGER	10000
10	PRESIDENT	15000
20	ANALYST	8000
20	MANAGER	13000
20	SALESMAN	4000
30	MANAGER	8000
30	SALESMAN	5000

4.2.1 ROLLUP Operator

The **ROLLUP** operator can be used to add one or more summary rows to a result set that uses grouping and aggregates. A summary is provided for each aggregate column included in the select list. All other columns, except the ones that identify which group is being summarized, are assigned null values. It also adds a summary row to the end of the result set that summarizes the entire result set.

- Display the amount of salary being paid jobwise for each department.

```
SELECT deptno, job, SUM(sal) FROM emp

GROUP BY ROLLUP (deptno, job)

ORDER BY deptno, job;
```

DEPTNO	**JOB**	**SUM(SAL)**
10	CLERK	7000
10	MANAGER	10000
10	PRESIDENT	15000
10	(null)	32000
20	ANALYST	8000
20	MANAGER	13000
20	SALESMAN	4000
20	(null)	25000
30	MANAGER	8000
30	SALESMAN	5000
30	(null)	13000
(null)	(null)	70000

4.2.2 CUBE Operator

The **CUBE** operator is similar to the ROLLUP operator, except it adds summary rows for every combination of groups specified in the GROUP BY clause. It also adds a summary row to the end result set that summarizes the entire result set.

(14) Display the amount of salary being paid jobwise for each department.

```
SELECT deptno, job, SUM(sal) FROM emp

GROUP BY CUBE (deptno, job)

ORDER BY deptno, job;
```

DEPTNO	**JOB**	**SUM(SAL)**
10	CLERK	7000
10	MANAGER	10000
10	PRESIDENT	15000
10	(null)	32000
20	ANALYST	8000
20	MANAGER	13000

20	SALESMAN	4000
20	(null)	25000
30	MANAGER	8000
30	SALESMAN	5000
30	(null)	13000
(null)	ANALYST	8000
(null)	CLERK	7000
(null)	MANAGER	31000
(null)	PRESIDENT	15000
(null)	SALESMAN	9000
(null)	(null)	70000

4.2.3 GROUPING Function

Using the ROLLUP and CUBE operator introduces a null value to the column in a summary row that hasn't been summarized. If you want to assign a value other than null to these columns, you can do it by using the **GROUPING** function. The GROUPING function determines when a null value is assigned to a column as a result of the ROLLUP or CUBE operator. The column named in this function must be one of the columns named in the GROUP BY clause.

If a null value is assigned to the specified column as a result of the ROLLUP or CUBE operator, the GROUPING function returns a value of 1. Otherwise it returns a value of 0.

- Display the amount of salary being paid jobwise for each department.

```
SELECT
        CASE
        WHEN GROUPING (deptno) = 1 THEN '=========='
                ELSE deptno
        END AS dept_no,
        CASE
                WHEN GROUPING (job) = 1 THEN '=========='
                ELSE job
        END AS job,
        SUM(sal)
FROM emp
GROUP BY ROLLUP (deptno, job)
ORDER BY deptno, job;
```

DEPTNO	JOB	SUM(SAL)
10	CLERK	7000
10	MANAGER	10000
10	PRESIDENT	15000
10	==========	32000
20	ANALYST	8000
20	MANAGER	13000
20	SALESMAN	4000
20	==========	25000
30	MANAGER	8000
30	SALESMAN	5000
30	==========	13000
==========	==========	70000

4.3 SET OPERATORS

Set operators combine the results of two component queries into a single result. Queries containing set operators are called **compound queries**. Like joins, set operators combine data from two or more tables but there is a big difference. Joins try to combine columns from the base tables, however, set operators combine rows from two or more result sets.

Generally **SET operations** are applied on multiple SELECT statements. The records returned by each SELECT statement are treated as a SET of values and the final result is obtained depending on the SET operator used. There are four SET operators available:

- UNION
- UNION ALL
- INTERSECT
- MINUS

Conditions for SET operations

Two SELECT statements can be combined into a compund query by a SET operation if they satisfy the following conditions:

- The result set of both the queries must have the same number of columns.
- The data type of each column in the first result set must match with the data types of the columns of the second result set.

Restrictions on SET operations

- The column names for the resultant data set will come from the first query.
- If you want to use the ORDER BY clause in the query involving SET operations, you must place the ORDER BY clause only once at the end of the compound query. The component queries can't have individual ORDER BY clauses.

Syntax:

```
<query 1>

[SET OPERATOR]

<query 2>
```

Assume two tables "student" and "student_details" as shown below.

CLASS	ROLLNO	NAME
201	1	Satish
201	2	Rashmi
205	3	Rishi
205	4	anil

ROLLNO	SUBJECT	MARKS
1	Maths	12
1	Physics	23
2	Maths	34
3	Maths	35

4.3.1 INTERSECT Operator

Returns only those rows which are common to both queries.

- Display the rollnos of students that are in both the student and student_details table.

```
SELECT rollno FROM student

INTERSECT

SELECT rollno FROM student_details;
```

ROLLNO
1
2
3

- Display the details of students from the student table for which marks have been entered in the student_details table.

```
SELECT * FROM student

WHERE rollno IN

(

        SELECT rollno FROM student

        INTERSECT

        SELECT rollno FROM student_details

);
```

CLASS	ROLLNO	NAME
201	1	Satish
201	2	Rashmi
205	3	Rishi

4.3.2 UNION Operator

Returns the values which exist in either of the two queries. By default, a UNION eliminates duplicate rows.

10. Display all the rollnos which exist either in the student or in the student_details table.

```
SELECT rollno FROM student

UNION

SELECT rollno FROM student_details;
```

ROLLNO
1
2
3
4

4.3.3 UNION ALL Operator

Returns the values which exist in either of the two queries. The **UNION** operator has an additional clause ALL which displays the duplicate values also.

Display all the rollnos which exist either in the student or in the student_details table.

```
SELECT rollno FROM student

UNION ALL

SELECT rollno FROM student_details;
```

ROLLNO
1
2
3
4
1
1
2
3

Using ORDER BY clause

In order to arrange the final result of any SET operator, we can use the ORDER BY clause at the end of the last SELECT statement.

- Display all the rollnos which exist either in the student or in the student_details table.

```
SELECT rollno FROM student

UNION ALL

SELECT rollno FROM student_details

ORDER BY rollno;
```

ROLLNO
1
1
1
2
2
3
3
4

4.3.4 MINUS Operator

The MINUS operator takes the result set of one SELECT statement, and removes those rows that are also returned by a second SELECT statement. It removes the records that are common to both the queries and displays the remaining records from the query 1.

19. Display the rollnos whose marks have not been entered in the student_details table.

```
SELECT rollno FROM student

MINUS

SELECT rollno FROM student_details;
```

ROLLNO
4

4.4 DATETIME FUNCTIONS

We saw a brief overview of DateTime functions in chapter 2. In this section we discuss about these functions in greater detail.

Datetime functions operate on date (DATE), timestamp (TIMESTAMP, TIMESTAMP WITH TIME ZONE, and TIMESTAMP WITH LOCAL TIME ZONE), and interval (INTERVAL DAY TO SECOND, INTERVAL YEAR TO MONTH) values.

Two operators, plus and minus signs, can be used to work with dates. Summary of these operators is provided in the table below.

Operator	**Description**
+	Adds the specified number of days to a date
-	Subtracts the specified number of days from a date. Or, subtracts one date from another and returns the number of days between the two dates.

The table given below gives the syntax of commonly used date time functions and also their explanation. If you study the summaries and examples of these functions, you shouldn't have much trouble using them.

Function	**Description**
SYSDATE	Returns the current local date and time based on the operating system's clock
CURRENT_DATE	Returns the local date and time adjusted for the current session time zone

ROUND (date [, date_format])	Returns the date rounded to the unit specified by the date format. If the format is omitted, rounds to the nearest day.
TRUNC (date [, date_format])	Works like the ROUND function but truncates the date.
MONTHS_BETWEEN (date1, date2)	Returns the number of months between date1 and date2.
ADD_MONTHS (date, integer_months)	Adds the specified number of months to the specified date and returns the resulting date.
LAST_DAY (date)	Returns the date for the last day of the month for the specified date.
NEXT_DAY (date, day_of_week)	Returns the date for the next day of the week that comes after the specified date.

Examples that use date/time functions

Example	**Result**
SYSDATE	19-OCT-09 04:23:36 PM
ROUND (SYSDATE)	20-OCT-09 12:00:00 AM
ROUND (SYSDATE, ’MI’)	19-OCT-09 04:24:00 PM
TRUNC (SYSDATE, ’MI’)	19-OCT-09 04:23:00 PM
MONTHS_BETWEEN (’15-SEP-08’, ’01-AUG-08’)	1.45161290.........
ADD_MONTHS (’19-OCT-09’, -1)	19-SEP-09

LAST_DAY ('15-FEB-09')	28-FEB-09
NEXT_DAY ('15-AUG-08', 'THURS')	21-AUG-08
SYSDATE – 1	18-OCT-09

For Example, let us find out the number of years for various employees who served the company. (refer to the emp table)

SELECT empno, ename, hiredate, ROUND((sysdate – hiredate)/365) no_of_years

FROM emp;

EMPNO	**ENAME**	**HIREDATE**	**NO_OF_YEARS**
111	Satish	19-DEC-2008	01
222	Rashmi	01-JAN-1987	22
333	Rishi	05-JUN-1976	33
444	Anil	16-APR-1967	42
666	Nilesh	20-MAY-1987	22
777	Ruchi	11-JUN-2000	9

Note: The employees whose hire date is NULL are not listed. Since the ROUND function is used without any specified format, rounding is done for the next year (i.e. for the first row the time between hiredate (19/12/08) and sysdate (19/10/2009) is 10 months).

4.4.1 Parsing Date and Time

TO_CHAR function can be used to return various parts of a DATE value as a string. To do that you need to specify the appropriate date format element for the part of the DATE value that you want to return.

Example	Result
TO_CHAR (SYSDATE, ’DD-MON-RR HH:MI:SS’)	19-OCT-09 04:23:36 PM
TO_CHAR (SYSDATE, ’YEAR’)	TWO THOUSAND NINE
TO_CHAR (SYSDATE, ’YYYY’)	2009
TO_CHAR (SYSDATE, ’YY’)	09
TO_CHAR (SYSDATE, ’MONTH’)	OCTOBER
TO_CHAR (SYSDATE, ’MON’)	OCT
TO_CHAR (SYSDATE, ’MM’)	10
TO_CHAR (SYSDATE, ’DAY’)	MONDAY
TO_CHAR (SYSDATE, ’DY’)	MON
TO_CHAR (SYSDATE, ’DD’)	19
TO_CHAR (SYSDATE, ’HH24’)	16
TO_CHAR (SYSDATE, ’HH’)	04
TO_CHAR (SYSDATE, ’MI’)	23
TO_CHAR (SYSDATE, ’Q’)	4

4.5 CONTROLLING USER ACCESS

Oracle is a multi-user RDBMS and provides a secure environment such that the objects owned by a user are by default not accessible to the other users. It provides the facility that the owner of an object can grant various permissions to other database users as per requirements.

Authorization includes primarily two processes:

- Permitting only certain users to access, process, or alter data.

- Applying varying limitations on user access or actions. The limitations placed on (or removed from) users can apply to objects such as schemas, tables, or rows or to resources such as time (CPU, connect, or idle times).

A user **privilege** is the right to run a particular type of SQL statement, or the right to access an object that belongs to another user, run a PL/SQL package, and so on.

Roles are created by users (usually administrators) to group together privileges or other roles. They are a way to facilitate the granting of multiple privileges or roles to users.

There are two types of privileges: system privileges and object privileges.

1. System privileges:

A **system privilege** is the right to perform a particular action or to perform an action on any schema objects of a particular type. For example, the privileges to create tablespaces and to delete the rows of any table in a database are system privileges.

Some of the most common system privileges are:

- CREATE / ALTER / DROP USER
- CREATE SESSION
- CREATE / ALTER / DROP TABLE
- CREATE VIEWS
- CREATE PROCEDURE
- CREATE SEQUENCE
- CREATE PUBLIC SYNONYM

2. Object Privileges:

An **object privilege** is a right that you grant to a user on a database objects like tables, views, sequences, packages, procedures. Some examples of object privileges include the right to:

- Use an edition
- Update a table
- Select rows from another user’s table
- Execute a stored procedure of another user

3. What a user can grant?

A user can grant privileges on any object he/she owns. Different objects have different permissions to be assigned to other users as specified in the table below.

Object	Privileges
Table	SELECT, INSERT, UPDATE, DELETE, ALTER, INDEX
View, Materialized Views	SELECT, INSERT, UPDATE, DELETE
Sequence	SELECT, ALTER
Functions, Procedures, Packages	EXECUTE
Index	EXECUTE

4.5.4 GRANT/REVOKE PRIVILEGES

The Data Control Language commands are used to enforce database security in a multiple user database environment. The Data Control Language (DCL) authorizes users and groups of users to access and manipulate data. Its two main statements are:

13. GRANT: authorizes one or more users to perform an operation or a set of operations on an object.
14. REVOKE: eliminates a grant, which may be the default grant.

4.5.4.1 GRANT COMMAND

The **GRANT** command is used to grant system and object privileges to a role or a user.

Granting System Privileges

You can grant system privileges to users and roles. If you grant system privileges to roles, then you can use the roles to exercise system privileges.

The **syntax** of GRANT statement for system privileges

```
GRANT <system_privilege>

TO <user_or_role>

[WITH ADMIN OPTION]
```

The **WITH ADMIN OPTION** clause allows the user or role to grant the specified system privileges to other users or roles.

For example,

- A statement that grants a system privilege to a role

GRANT CREATE SESSION TO ap_user;

- A statement that grants a system privilege to a role with the admin option

GRANT CREATE SESSION TO ap_developer WITH ADMIN OPTION;

Granting Object Privileges

Each type of object has different privileges associated with it. Object privileges can be granted to users and roles. If you grant object privileges to roles, then you can make the privileges selectively available.

The **syntax** of GRANT statement for object privileges

GRANT <object_privilege>

ON <[schema_name.]object_name [(column [,.....])]>

TO <user_or_role>

[WITH GRANT OPTION]

The **WITH GRANT OPTION** clause allows the user or role to grant the specified object privileges to other users or roles.

Consider the following examples,

- A statement that grants an object privilege to a role

GRANT SELECT ON emp TO ap_user;

- A statement that grants all object privileges to a role with the grant option

GRANT SELECT, INSERT, UPDATE, DELETE ON emp TO ap_developer WITH GRANT OPTION;

4.5.4.2 REVOKE COMMAND

The **REVOKE** command is used to revoke system or object privileges from a role or user.

Revoking System Privileges

You can revoke system privileges from roles or users.

The **syntax** of REVOKE statement for system privileges

REVOKE <system_privilege>

FROM <user_or_role>

Consider the following examples,

- A statement that revokes a system privilege from a role

REVOKE DROP ANY VIEW FROM ap_user;

Revoking Object Privileges

Each type of object has different privileges associated with it. Object privileges can be revoked from users and roles.

The **syntax** of REVOKE statement for object privileges

REVOKE [GRANT OPTION FOR] <object_privilege>

ON <[schema_name.]object_name [(column [,.....])]>

FROM <user_or_role>

[RESTRICT | CASCADE]

The **GRANT OPTION FOR** clause allows the user or role to revoke the specified object privileges from other users or roles.

The **RESTRICT** clause revokes all privileges for the user.

The **CASCADE** clause revokes all privileges for the user and given by the user to other users.

Consider the following examples,

- A statement that revokes an object privilege from a role

REVOKE SELECT ON emp FROM ap_user;

- A statement that revokes selected object privileges from a role with the grant option

REVOKE GRANT OPTION FOR SELECT, INSERT, ON emp FROM ap_developer RESTRICT;

NOTE:

You can specify **ALL [PRIVILEGES]** to grant or revoke all available object privileges for an object. **ALL** is not a privilege; rather, it is a shortcut, or a way of granting or revoking all object privileges with one **GRANT** and **REVOKE** statement. If all object privileges are granted using the ALL shortcut, then individual privileges can still be revoked.

Similarly, you can revoke all individually granted privileges by specifying ALL.

Consider the following examples,

- A statement that grants all object privileges to a role.

GRANT ALL ON emp TO ap_developer;

- A statement that revokes all object privilege from a role

REVOKE ALL ON emp FROM ap_user;

4.6 SUMMARY

- The ROLLUP operator can be used to add one or more summary rows to a result set that uses grouping and aggregates.
- The CUBE operator is similar to the ROLLUP operator, except it adds summary rows for every combination of groups specified in the GROUP BY clause.
- The GROUPING function determines when a null value is assigned to a column as a result of the ROLLUP or CUBE operator.
- Set operators combine the results of two component queries into a single result. Queries containing set operators are called compound queries.
- The summary of various SET operators is as follows:

Operator	Description
UNION ALL	Combines the results of two SELECT statements into one result set.
UNION	Combines the results of two SELECT statements into one result set, and then eliminates any duplicate rows from that result set.
MINUS	Takes the result set of one SELECT statement, and removes those rows that are also returned by the another SELECT statement.
INTERSECT	Returns only those rows that are returned by each of the two SELECT statements.

- Datetime functions operate on date, timestamp, and interval values.
- TO_CHAR function can be used to return various parts of a DATE value as a string.
- A user privilege is the right to run a particular type of SQL statement, or the right to access an object that belongs to another user, run a PL/SQL package, and so on.
- There are two types of privileges: system privileges and object privileges.
- A system privilege is the right to perform a particular action or to perform an action on any schema objects of a particular type.
- An object privilege is a right that you grant to a user on a database objects like tables, views, sequences, packages, procedures.
- The GRANT command is used to grant system and object privileges to a role or a user.
- The REVOKE command is used to revoke system or object privileges from a role or user.

4.7 REVIEW QUESTIONS

1. Explain the ROLLUP and CUBE operator with examples.
2. What are SET operators? List and explain the different types of SET operators.
3. Using DateTime functions, how to calculate age from date of birth?
4. What is a privilege? Explain the different types of privileges.
5. Explain the GRANT command in detail with examples.
6. Explain the REVOKE command in detail with examples.

4.8 LAB ASSIGNMENT

1. Create a CUSTOMER table with the following columns and constraints

Column name	Data type	Size	Constraint
CUSTOMER_ID	CHAR	6	PRIMARY KEY. MUST BEGIN WITH 'C'
CUSTOMER_NAME	VARCHAR2	20	NOT NULL
ADDRESS	VARCHAR2	20	UNIQUE
CITY	VARCHAR2	20	

PINCODE	NUMBER	6	
STATE	VARCHAR2	20	
BALANCE_DUE	NUMBER	8,2	

2. Create a PRODUCT table with the following columns and constraints

Column name	Data type	Size	Constraint
PRODUCT_CODE	CHAR	6	PRIMARY KEY
PRODUCT_NAME	VARCHAR2	20	UNIQUE
QTY_AVAIL	NUMBER	5	
COST_PRICE	NUMBER	8,2	
SELLING_PRICE	NUMBER	8,2	

3. Create a ORDER table with the following columns and constraints

Column name	Data type	Size	Constraint
ORDER_NO	CHAR	6	
ORDER_DATE	TIMESTAMP		
CUSTOMER_ID	CHAR	6	
PRODUCT_CODE	CHAR	6	
QUANTITY	NUMBER	5	

PRIMARY KEY = ORDER_NO + ORDER_DATE + CUSTOMER_ID + PRODUCT_CODE

4. Insert 5-10 records in all the tables.
5. Apply UNIQUE constraint on CUSTOMER_ID+PRODUCT_CODE on the ORDER table.
6. Define a foreign key on CUSTOMER_ID of ORDER table referring to CUSTOMER_ID of CUSTOMER table.
7. Define a foreign key on PRODUCT_CODE of ORDER table referring to PRODUCT_CODE of PRODUCT table.
8. Find the total number of orders and the customerwise number of orders from the ORDER table.
9. Display the orders placed during the year '2008'.
10. What would be the date 10 days from today.
11. 28/02/2010 would fall on which day (Mon, Tue, ...)?
12. List the customer codes that have placed orders using SET operators.
13. Display the customer codes that have not placed any order using SET operators.
14. Display the customer codes that have placed some order using SET operators.

15. Permit the user “Ram” to be able to INSERT and DELETE commands in the CUSTOMER table.
16. Grant INSERT permission to the user “Ravi” such that he can further grant the INSERT permission to other users for the CUSTOMER table.
17. Withdraw the DELETE permission from “Ram”.
18. Give the UPDATE permission on ADDRESS and STATE column of the CUSTOMER table to “Kishan”.

4.9 BIBLIOGRAPHY, REFERENCES AND FURTHER READING

- Database Management Systems, Third Edition by RamaKrishnan, Gehre. McGraw Hill
- Database System Concepts, Fifth Edition by Silberschatz, Korth, Sudarshan. McGraw Hill
- Murach’s Oracle SQL and PL/SQL by Joel Murach. Shroff Publishers & Distributors
- Oracle Database 11g by Satish Asnani. PHI Learning Private Limited
- Oracle 11g: PL/SQL Reference Oracle Press.
- Expert Oracle PL/SQL, By: Ron Hardman,Michael McLaughlin, Tata McGraw-Hill
- SQL, PL/SQL The programming language of Oracle, Bayross Ivan, BPB Publications
- Fundamentals of Database Systems, Elmasri Ramez and Navathe B. Shamkant, Pearson

4.10 ONLINE REFERENCES

Wikipedia Link

http://en.wikipedia.org/wiki/SQL

Oracle Database PL/SQL language Reference 11g Release 2 (11.2), part number E25519-05

http://docs.oracle.com/cd/E11882_01/appdev.920/a96590/adg09dyn.htm

5

ADVANCED SUBQUERIES

Unit Structure

1. OBJECTIVES

At the end of this chapter you will be able to,

15. Understand Scalar and Correlated Subqueries
16. Write and Execute Multiple Column Subqueries
17. Understand WITH clause
18. Understand Hierarchical queries

5.1 INTRODUCTION

In the previous chapters we learnt that, queries can be nested so that the results of one query can be used in another query via a relational operator or aggregation function. This type of nested query is known as a subquery. In this chapter we will discuss more about the specifics of using subqueries.

To illustrate the concepts in this chapter, assume the tables "**EMP**" and "**EMP_DETAILS**" having the following records.

Table EMP:

EMPID	DESIGNATION	DEPT	SALARY
1001	Manager	Finance	50000
1002	Executive	Finance	25000
1003	Senior Executive	Finance	35000
1004	Manager	HR	20000
1005	Executive	HR	20000
1006	Senior Executive	HR	30000
1007	Manager	Admin	55000
1008	Executive	Finance	25000
1009	Executive	Finance	25000
1010	Executive	HR	25000

Table EMP_DETAILS:

EMPID	ENAME	AGE
1001	Rajesh	23
1002	Tejal	25

1003	Himesh	35
1004	Himali	37
1005	Rehan	40
1006	Kiran	28
1007	Ashima	21
1008	Vikram	31
1009	Ridhi	26

5.2 MULTIPLE COLUMN SUBQUERIES

Multiple column subquery returns the rows on the basis of matching of a pair of given columns for a given row. It first selects the row on the basis of the WHERE clause and then finds the other rows on the basis of a matching pair of columns of the particular row. There are two types of comparison in multiple column subqueries.

Pair wise: In pair wise comparison we search both the columns match in the same subquery, e.g. "**WHERE** (column1, column2) **IN** (subquery1)"

Non Pair wise: In non pair wise comparison we search the columns in separate subqueries, e.g. "**WHERE** (column1) **IN** (subquery1) **AND** (column2) **IN** (subquery2)".

- List the empid of employees who have the same salary and designation as the employee having empid 1009.

```
SELECT DISTINCT empid FROM emp

WHERE (designation, salary)

IN (SELECT designation, salary FROM emp WHERE empid =
1009)

ORDER BY empid;
```

Or

```
SELECT DISTINCT empid FROM emp

WHERE ((designation) IN (SELECT designation, salary FROM
emp WHERE empid = 1009)) AND ((salary) IN (SELECT
designation, salary FROM emp WHERE empid = 1009))

ORDER BY empid;
```

Both the queries above will give the same result set. The first query is the pair wise comparison while the second query is the non pair wise comparison. The output of the above query is:

EMPID
1002
1008
1009
1010

5.2.1 CODING SUBQUERIES IN THE FROM CLAUSE

A subquery can be coded in place of a table specification i.e. in the **FROM** clause. The results of the subquery are joined with another table. When you use a subquery in this way, it can return any number of rows and columns. This type of subquery, in the FROM clause of a SELECT statement, is referred to as an **inline view** since it works like a view that is temporarily created and stored in the memory. Inline views are most useful when you need to summarize the results of a summary query. When you create an inline view, you must assign an alias to it. Then, you can use the inline view within the outer query just as you would any other table.

For example,

- To find the department in which the maximum number of employees work.

  SELECT MAX (inview.total_emp)

  FROM (SELECT COUNT (empid) as total_emp FROM emp GROUP BY dept) inview;

MAX(INVIEW.TOTAL_EMP)
5

In the above query, we use the alias **'inview'** for the subquery that begins after the FROM clause. The subquery counts the number of employees that work in each department & the outer query takes the values from the subquery using the MAX function finds the maximum number of employees to give the result set.

3. SCALAR SUBQUERIES

A **scalar subquery** expression is a subquery that returns exactly one column value from one row. The value of the scalar subquery expression is the value of the select list item of the subquery. If the subquery returns 0 rows, then the value of the scalar subquery expression is NULL. If the subquery returns more than one row, then Oracle returns an error.

You can use a scalar subquery expression in most syntax that calls for an expression (*expr*).

For example,

- Get the name and age of the employee enjoying maximum salary.

```
SELECT ename, age FROM emp_details

WHERE empid = (SELECT empid FROM emp WHERE salary =

        (SELECT MAX(salary) FROM emp));
```

ENAME	**AGE**
Ashima	21

In the above example, the innermost scalar subquery gets executed first providing the maximum salary from the EMP table, then the empid of the employee is selected and given to the outer query to get the name and age of the employee.

5.4 CORRELATED SUBQUERY

A **correlated subquery** is a subquery that is executed once for each row processed by the outer query. It's similar to using a loop to do repetitive processing in a procedural programming. In contrast, a noncorrelated subquery is executed only once. In correlated subquery, the outer query executes first and the inner query will execute second. Each subquery is executed once for every row of the outer query.

For example,

- Get the list of employees whose salary are higher than or equal to the average salary of their respective departments.

```
SELECT e1.empid, e1.dept FROM emp e1
WHERE salary >= (SELECT AVG(salary) FROM emp e2 GROUP
BY e2.dept
        HAVING e1.empid = e2.empid);
```

EMPID	DEPT
1001	Finance
1003	Finance
1006	HR
1007	Admin

To run the inner query we need to know the "dept" of the employee selected in the outer query. For each and every department value coming from the outer query the average salary of the department is calculated in the inner query and compared with the salary of the employee in the outer query.

5.5 WITH CLAUSE (SUBQUERY FACTORING CLAUSE)

The WITH *query_name* clause lets you assign a name to a subquery block. You can then reference the subquery block multiple places in the query by specifying *query_name*. SQL optimizes the query by treating the query name as either an inline view or as a temporary table.

Syntax:

WITH query_name ([c_alias [, c_alias]...]) **AS** (subquery)

[, query_name ([c_alias [, c_alias]...]) **AS** (subquery)]...

The column aliases following the *query_name* and the set operators separating multiple subqueries in the AS clause are valid. You can specify this clause in any top-level SELECT statement and in most types of subqueries. The query name is visible to the main query and to all subsequent subqueries. To code multiple subquery factoring clauses, separate them with commas. Then each clause can refer to itself and any previously defined subquery factoring clauses in the same WITH clause.

5.5.1 FUNCTIONS OF THE WITH CLAUSE

(15) A named query can be referenced any number of times.

(16) Any number of named queries can be created.

(17) Named queries can reference other named queries that came before them and even correlate to previous named queries.

(18) The scope of the WITH clause is local to the SELECT in which they are defined.

Consider the following examples,

- Get the empid of the employee who works in the HR department and gets the maximum salary.

  WITH hr_sal AS

  (SELECT empid, salary FROM emp WHERE dept = 'HR')

  SELECT MAX (salary) FROM hr_sal;

MAX (SALARY)
30000

- Get the average age of employees working in the "Finance" department.

  WITH fin_age AS

  (SELECT empid, salary FROM emp WHERE dept = 'Finance'

  SELECT AVG (age) FROM emp_details

  WHERE empid IN (SELECT empid FROM fin_age);

5.6 HIERARCHICAL QUERIES

A **hierarchical query** loops through a result set and returns rows in a hierarchical sequence. If a table contains hierarchical data, then you can select rows in a hierarchical order using the hierarchical query clause.

Syntax:

SELECT select_list

FROM table_name

[**WHERE** search_condition]

START WITH row_specification

CONNECT BY PRIOR connect_expr

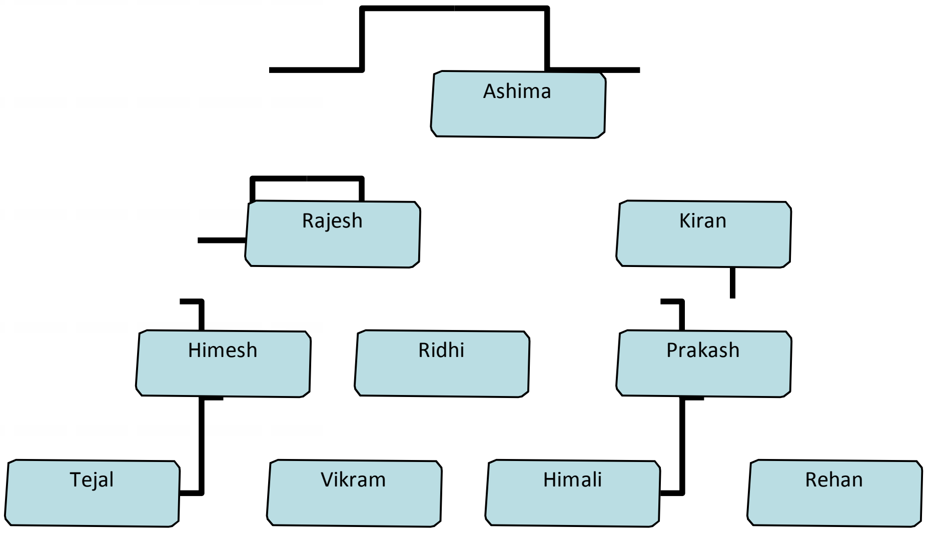
SELECT statements that contain hierarchical queries can contain the **LEVEL** pseudocolumn in the select list. LEVEL returns the value 1 for a root node, 2 for a child node of a root node, 3 for a grandchild, and so on. The number of levels returned by a hierarchical query may be limited by available user memory.

START WITH Clause – used to specify a condition that identifies the row(s) to be used as the root(s) of a hierarchical query.

CONNECT BY Clause – followed by the **PRIOR** keyword is used to specify a condition that identifies the relationship between parent rows and child rows of the hierarchy.

Assume the records in the EMP_NEW table

EMPID	ENAME	DESIGNATION	DEPT	SALARY	MGRID
1001	Rajesh	Manager	Finance	50000	1007
1002	Tejal	Executive	Finance	25000	1003
1003	Himesh	Senior Executive	Finance	35000	1001
1004	Himali	Manager	HR	20000	1010
1005	Rehan	Executive	HR	20000	1010
1006	Kiran	Senior Executive	HR	30000	1007
1007	Ashima	President	Admin	55000	-
1008	Vikram	Executive	Finance	25000	1003
1009	Ridhi	Executive	Finance	25000	1001
1010	Prakash	Executive	HR	25000	1006



A query that returns hierarchical data

SELECT LEVEL, empid, ename, mgrid FROM emp_new

START WITH ename = “Ashima”

CONNECT BY PRIOR empid = mgrid

ORDER BY LEVEL, empid;

LEVEL	EMPID	ENAME	MGRID
1	1007	Ashima	-
2	1001	Rajesh	1007
2	1006	Kiran	1007
3	1003	Himesh	1001
3	1009	Ridhi	1001
3	1010	Prakash	1006
4	1002	Tejal	1003
4	1004	Himali	1010
4	1005	Rehan	1010
4	1008	Vikram	1003

The EMP_NEW table uses the MGR column to identify the manager for each employee. Here, Ashima is the top level manager since she doesn’t have a manager. Rajesh and Kiran report to Ashima and so on.

The hierarchical query uses the LEVEL pseudo-column to return a column that identifies the level of the employee within the hierarchy. In addition, this query uses the LEVEL pseudo-column in the ORDER BY clause to sort by this column.

After the FROM clause, this query uses the START WITH clause to identify the row to be used as the root of the hierarchy. Finally the CONNECT BY clause specifies the condition that identifies the relationship between the parent rows and the child rows.

5.7 SUMMARY

- **Multiple column subquery** returns the rows on the basis of matching of a pair of given columns for a given row. It first selects the row on the basis of the WHERE clause and then finds the other rows on the basis of a matching pair of columns of the particular row.
- There are two types of comparison in multiple column subqueries - **Pair-Wise and Non Pair-wise**.
- A **subquery** can be coded in place of a table specification i.e. in the **FROM** clause. The results of the subquery are joined with another table.
- A **scalar subquery** expression is a subquery that returns exactly one column value from one row.
- A **correlated subquery** is a subquery that is executed once for each row processed by the outer query. It’s similar to using a loop to do repetitive processing in a procedural programming.
- The **WITH query_name** clause lets you assign a name to a subquery block.
- To code multiple subquery factoring clauses, separate them with commas. Then each clause can refer to itself and any previously defined subquery factoring clauses in the same WITH clause.
- If a table contains hierarchical data, then you can select rows in a hierarchical order using the **hierarchical query** clause.

5.8 REVIEW QUESTIONS

- Explain multiple column subqueries with suitable examples.
- What is an inline view? When can you use it? Explain with example.
- What is a scalar subquery? Explain with suitable example.
- What is a correlated subquery? Explain with suitable example.

- Explain the WITH clause. Why should you use the WITH clause?
- What is a hierarchical query? Illustrate with the help of an example.

5.9 LAB ASSIGNMENT

1. Create a CUSTOMER table with the following columns and constraints

Column name	Data type	Size	Constraint
CUSTOMER_ID	CHAR	6	PRIMARY KEY. MUST BEGIN WITH 'C'
CUSTOMER_NAME	VARCHAR2	20	NOT NULL
ADDRESS	VARCHAR2	20	UNIQUE
CITY	VARCHAR2	20	
PINCODE	NUMBER	6	
STATE	VARCHAR2	20	
BALANCE_DUE	NUMBER	8,2	

2. Create a PRODUCT table with the following columns and constraints

Column name	Data type	Size	Constraint
PRODUCT_CODE	CHAR	6	PRIMARY KEY
PRODUCT_NAME	VARCHAR2	20	UNIQUE
QTY_AVAIL	NUMBER	5	
COST_PRICE	NUMBER	8,2	
SELLING_PRICE	NUMBER	8,2	

3. Create a ORDER table with the following columns and constraints

Column name	Data type	Size	Constraint
ORDER_NO	CHAR	6	
ORDER_DATE	TIMESTAMP		
CUSTOMER_ID	CHAR	6	
PRODUCT_CODE	CHAR	6	
QUANTITY	NUMBER	5	

PRIMARY KEY = ORDER_NO + ORDER_DATE + CUSTOMER_ID + PRODUCT_CODE

- Insert 5-10 records in all the tables.
- Apply UNIQUE constraint on CUSTOMER_ID+PRODUCT_CODE on the ORDER table.
- Define a foreign key on CUSTOMER_ID of ORDER table referring to CUSTOMER_ID of CUSTOMER table.
- Define a foreign key on PRODUCT_CODE of ORDER table referring to PRODUCT_CODE of PRODUCT table.
- List the customer names in the order of decreasing quantity ordered.
- Calculate the average selling price of all products.
- List the customer names for which we have orders in hand.
- List the details of all products whose price is less than average price of the products.
- List the customers who have ordered for the same products.
- Get the name, price and quantity in stock of the costliest product.

10. BIBLIOGRAPHY, REFERENCES AND FURTHER READING

- Database Management Systems, Third Edition by RamaKrishnan, Gehre. McGraw Hill
- Database System Concepts, Fifth Edition by Silberschatz, Korth, Sudarshan. McGraw Hill
- Murach's Oracle SQL and PL/SQL by Joel Murach. Shroff Publishers & Distributors

- Oracle Database 11g by Satish Asnani. PHI Learning Private Limited
- Oracle 11g: PL/SQL Reference Oracle Press.
- Expert Oracle PL/SQL, By: Ron Hardman,Michael McLaughlin, Tata McGraw-Hill
- SQL, PL/SQL The programming language of Oracle, Bayross Ivan, BPB Publications
- Fundamentals of Database Systems, Elmasri Ramez and Navathe B. Shamkant, Pearson

5.11 ONLINE REFERENCES

Wikipedia Link

http://en.wikipedia.org/wiki/SQL

Oracle Database PL/SQL language Reference 11g Release 2 (11.2), part number E25519-05

http://docs.oracle.com/cd/E11882_01/appdev.920/a96590/adg09dyn.htm

Thank You

educlash.com
Just another Way To Learn