Laboratory Manual

ON

DATA BASE MANAGEMENT SYSTEM LAB

(For 4th Semester CSE/IT)

Prepared by:

Sri Ramesh Chandra Sahoo
Senior Lecturer (CSE&IT)
UCP Engineering School
Berhampur.

Smt Reetanjali Panda
Lecturer(CSE&IT)
UCP Engineering School
Berhampur.

GETTING STARTED WITH ORACLE

What is a database?

A database is an organized collection of structured data stored electronically in a computer system. In the 1970s, Dr. E.F.Codd, a computer scientist, invented the relational model for database management. The relational model deals with many issues caused by the flat file model. According to his model, data is organized in entities and attributes, instead of combining everything in a single structure. By the way, we often refer the entities as tables, records as rows and fields as columns.

The relational model is better than the flat file model because it removes the duplicate data e.g. if you put employee and contact information on the same file. The employee, who has more than one contact, will appear in multiple rows.

The Relational Database Management System, or **RDBMS** in short, manages relational data. Oracle Database is an RDBMS with the largest market share.

Oracle Database features

Oracle Database allows you to quickly and safely store and retrieve data. Here are the integration benefits of the Oracle Database:

- Oracle Database is cross-platform. It can run on various hardware across operating systems including Windows Server, Unix, and various distributions of GNU/Linux.

- Oracle Database has its networking stack that allows application from a different platform to communicate with the Oracle Database smoothly. For example, applications running on Windows can connect to the Oracle Database running on Unix.
- ACID-compliant – Oracle is ACID-compliant Database that helps maintain data integrity and reliability.
- Commitment to open technologies – Oracle is one of the first Database that supported GNU/Linux in the late 1990s before GNU/Linux become a commerce product. It has been supporting this open platform since then.

Oracle Database has several structural features that make it popular:

- Logical data structure – Oracle uses the logical data structure to store data so that you can interact with the database without knowing where the data is stored physically.
- Partitioning – is a high-performance feature that allows you to divide a large table into different pieces and store each piece across storage devices.
- Memory caching – the memory caching architecture allows you to scale up a very large database that still can perform at a high speed.
- Data Dictionary is a set of internal tables and views that support administer Oracle Database more effectively.
- Backup and recovery – ensure the integrity of the data in case of system failure. Oracle includes a powerful tool called Recovery Manager (RMAN) – allows DBA to perform cold, hot, and incremental database backups and point-in-time recoveries.
- Clustering – Oracle Real Application Clusters (RAC) – Oracle enables high availability that enables the system is up and running without interruption of services in case one or more server in a cluster fails.

Oracle Data Types

In Oracle, every value has a data type which defines a set of characteristics for the value. These characteristics cause Oracle to treat values of one data type differently from values of another. For example, you can add values of the NUMBER data type, but not values of the RAW data type.

When you create a new table, you specify a data type for each of its columns. Similarly, when you create a new procedure, you specify a data type for each of its arguments. The data type defines the allowed values that each column or argument can store. For example, a DATE column cannot store a value of February 30, because this is not a valid date.

Oracle has a number of built-in data types illustrated in the following table:

Code	Data Type
1	VARCHAR2(size [BYTE | CHAR])
1	NVARCHAR2(size)
2	NUMBER[(precision [, scale]])
8	LONG
12	DATE
21	BINARY_FLOAT
22	BINARY_DOUBLE
23	RAW(size)
24	LONG RAW
69	ROWID

Code	Data Type
96	CHAR [(size [BYTE | CHAR])]
96	NCHAR[(size)]
112	CLOB
112	NCLOB
113	BLOB
114	BFILE

Each data type has a code managed internally by Oracle. To find the data type code of a value in a column, you use the DUMP() function.
Character data types

Character data types consist of CHAR, NCHAR, VARCHAR2, NVARCHAR2, and VARCHAR

The NCHAR and NVARCHAR2 data types are for storing Unicode character strings.

The fixed-length character data types are CHAR, NCHAR and the variable-length character data types are VARCHAR2, NVARCHAR2.

VARCHAR is the synonym of VARCHAR2. However, you should not use VARCHAR because Oracle may change its semantics in the future.

For character data types, you can specify their sizes either in bytes or characters.

Number data type

The NUMBER data type has precision **p** and scale **s**. The precision ranges from 1 to 38 while the scale range from -84 to 127.

If you don't specify the precision, the column can store values including fixed-point and floating-point numbers. The default value for the scale is zero.

Date data types : They are used to store date and time in a table. Default date data type is "dd-mon-yy". To view system's date and time we can use the SQL function called sysdate(). Oracle uses its own format to store date in a fixed length of 7 bytes each for century,month,day,year,hour,minute and second.

RAW and LONG RAW data types

The RAW and LONG RAW data types are for storing binary data or byte strings e.g., the content of documents, sound files, and video files.

The RAW data type can store up to 2000 bytes while the LONG RAW data type can store up to 2GB.

BFILE Datatype

BFILE data type stores a locator to a large binary file which locates outside the database. The locator consists of the directory and file names.

BLOB Datatype

BLOB stands for binary large object. You use the BLOB data type to store binary objects with the maximum size of (4 gigabytes – 1) * (database block size).

CLOB Datatype

CLOB stands for character large object. You use CLOB to store single-byte or multibyte characters with the maximum size is (4 gigabytes – 1) * (database block size).

Note that CLOB supports both fixed-with and variable-with character sets.

NCLOB Datatype

NCLOB is similar to CLOB except that it can store the Unicode characters.

Data Definition Language:

The Data Definition Language is used to create an object(table),alter the structure of an object and drop the object created.

1. Oracle CREATE TABLE

To create a new table in Oracle Database, we can use the CREATE TABLE statement. The following illustrates the basic syntax of the CREATE TABLE statement:

```
SQL>CREATE TABLE table_name (
   column_1 data_type column_constraint,
   column_2 data_type column_constraint,
   ...
   table_constraint
);
```

In this syntax:

- First, specify the table name on the CREATE TABLE clause.

- Second, list all columns of the table within the parentheses. In case a table has multiple columns, we need to separate them by commas (,). A column definition includes the column name followed by its data type e.g., NUMBER, VARCHAR2, and a column constraint such as NOT NULL, primary key, check.
- Third, add table constraints if applicable e.g., primary key, foreign key, check.

Oracle CREATE TABLE statement example

The following example shows how to create a new table named persons.

```
SQL>CREATE TABLE persons(
   person_id NUMBER,    first_name VARCHAR2(50) NOT NULL,
   last_name VARCHAR2(50) NOT NULL,
   PRIMARY KEY(person_id));
```

In this example, the persons table has three columns: person_id, first_name, and last_name.The data type of the person_id column is NUMBER. The first_name column has data type VARCHAR2 with the maximum length is 50. It means that we cannot insert a first name whose length is greater than 50 into the first_name column. Besides, the NOT NULL column constraint prevents the first_name column to have NULL values.The last_name column has the same characteristics as the first_name column.The PRIMARY KEY clause specifies the person_id column as the primary key column which is used for identifying the unique row in the persons table.

2.Oracle ALTER TABLE

To modify the structure of an existing table, we use the ALTER TABLE statement. The following illustrates the syntax:
SQL>ALTER TABLE table_name action;
In this statement:

- First, specify the table name which we want to modify.
- Second, indicate the action that you want to perform after the table name.

The ALTER TABLE statement allows you to:

- Add one or more columns
- Modify column definition
- Drop one or more columns

Let's see some examples to understand how each action works.

Oracle ALTER TABLE examples

Oracle ALTER TABLE ADD column examples

To add a new column to a table, we use the following syntax:

```
SQL>ALTER TABLE table_name
ADD column_name type constraint;
```

For example, the following statement adds a new column named birthdate to the persons table:

```
ALTER TABLE persons
ADD birthdate DATE NOT NULL;
```

If you view the persons table, you will see that the birthdate column is appended at the end of the column list:

```
DESC persons;

Name       Null     Type
---------- -------- ------------
PERSON_ID  NOT NULL NUMBER
FIRST_NAME NOT NULL VARCHAR2(50)
LAST_NAME  NOT NULL VARCHAR2(50)
BIRTHDATE  NOT NULL DATE
```

To add multiple columns to a table at the same time, you place the new columns inside the parenthesis as follows:

```
SQL>ALTER TABLE table_name
ADD (
   column_name type constraint,
   column_name type constraint,
   ...
);
```

```
SQL>ALTER TABLE persons
ADD (
   phone VARCHAR(20),
   email VARCHAR(100)
);
```

In this example, the statement added two new columns named phone and email to the persons table.

```
SQL>DESC persons

Name       Null     Type
---------- -------- -------------
PERSON_ID  NOT NULL NUMBER
FIRST_NAME NOT NULL VARCHAR2(50)
LAST_NAME  NOT NULL VARCHAR2(50)
BIRTHDATE  NOT NULL DATE
PHONE               VARCHAR2(20)
EMAIL               VARCHAR2(100)
```

Oracle ALTER TABLE MODIFY column examples

To modify the attributes of a column, we use the following syntax:

```
SQL>ALTER TABLE table_name MODIFY column_name type constraint;
```

For example, the following statement changes the birthdate column to a null-able column:

```
SQL>ALTER TABLE persons MODIFY birthdate DATE NULL;
```

Let's verify the persons table structure again:

```
SQL>DESC persons
```

```
Name       Null     Type
PERSON_ID  NOT NULL NUMBER
FIRST_NAME NOT NULL VARCHAR2(50)
LAST_NAME  NOT NULL VARCHAR2(50)
BIRTHDATE          DATE
PHONE              VARCHAR2(20)
EMAIL              VARCHAR2(100)
```

We can see, the birthdate became null-able.

To modify multiple columns, you use the following syntax:

```
SQL>ALTER TABLE table_name MODIFY ( column_1 type constraint, column_2 type constraint, ...);
```

For example, the following statement changes the phone and email column to NOT NULL columns and extends the length of the email column to 255 characters:

```
SQL> ALTER TABLE persons MODIFY( phone VARCHAR2(20) NOT NULL,email VARCHAR2(255) NOT NULL);
```

Let us verify the persons table structure again:

```
SQL>DESC persons;
```

```
Name       Null     Type
---------- -------- -------------
PERSON_ID  NOT NULL NUMBER
FIRST_NAME NOT NULL VARCHAR2(50)
LAST_NAME  NOT NULL VARCHAR2(50)
BIRTHDATE          DATE
PHONE      NOT NULL VARCHAR2(20)
EMAIL      NOT NULL VARCHAR2(255)
```

Oracle ALTER TABLE DROP COLUMN example

To remove an existing column from a table, we use the following syntax:

```
SQL>ALTER TABLE table_name DROP COLUMN column_name;
```

This statement deletes the column from the table structure and also the data stored in that column.

The following example removes the birthdate column from the persons table:

```
SQL> ALTER TABLE persons DROP COLUMN birthdate;
```

Viewing the persons table structure again, we will find that the birthdate column has been removed:

```
SQL> DESC persons;

Name       Null     Type
---------- -------- -------------
PERSON_ID  NOT NULL NUMBER
FIRST_NAME NOT NULL VARCHAR2(50)
LAST_NAME  NOT NULL VARCHAR2(50)
PHONE      NOT NULL VARCHAR2(20)
EMAIL      NOT NULL VARCHAR2(255)
```

To drop multiple columns at the same time, you use the syntax below:

```
SQL>ALTER TABLE table_name DROP (column_1,column_2,...);
```

For example, the following statement removes the phone and email columns from the persons table:

```
SQL>ALTER TABLE persons DROP( email, phone );
```

Let's check the persons table again:

```
SQL>DESC persons;


Name       Null     Type
---------- -------- ------------
PERSON_ID  NOT NULL NUMBER
FIRST_NAME NOT NULL VARCHAR2(50)
LAST_NAME  NOT NULL VARCHAR2(50)
```

3.Oracle DROP TABLE

To move a table to the recycle bin or remove it entirely from the database, we use the DROP TABLE statement:

```
SQL>DROP TABLE table_name;
```

In this statement:

Oracle DROP TABLE examples

The following CREATE TABLE statement creates persons table for the demonstration:

```
SQL>CREATE TABLE persons (
    person_id NUMBER,
    first_name VARCHAR2(50) NOT NULL,
    last_name VARCHAR2(50) NOT NULL,
    PRIMARY KEY(person_id)
);
```

The following example drops the persons table from the database:

```
SQL>DROP TABLE persons;
```

4.Oracle TRUNCATE TABLE:

Oracle introduced the TRUNCATE TABLE statement that allows us to delete all rows from a big table.

The following illustrates the syntax of the Oracle TRUNCATE TABLE statement:

```
SQL>TRUNCATE TABLE table_name;
```

By default, to remove all rows from a table, we specify the name of the table that we want to truncate in the TRUNCATE TABLE clause:
Example:

```
SQL>TRUNCATE TABLE persons;
```

5.Oracle RENAME Table

To rename a table, we use the following Oracle RENAME table statement as follows:

```
SQL>RENAME table_name TO new_name;
```

In the RENAME table statement:

- First, specify the name of the existing table which we want to rename.
- Second, specify the new table name. The new name must not be the same as another table in the same schema.

Note that we cannot roll back a RENAME statement once we executed it.

When we rename a table, Oracle automatically transfers indexes, constraints, and grants on the old table to the new one. In addition, it invalidates all objects that depend on the renamed table such as views, stored procedures, function, and synonyms.

Data Manipulation Language(DML)

These are used to query and manipulate existing objects like tables.

- **INSERT INTO/VALUES**
- This command is used for inserting values into the rows of a table (relation).
- Syntax: SQL>INSERT INTO table (column1 [, column2, column3 ...]) VALUES (value1 [, value2, value3 ...]);
- **For example:**
- SQL>INSERT INTO ucpes (Author, Subject) VALUES ("anonymous", "computers");
- **UPDATE/SET/WHERE**
- This command is used for updating or modifying the values of columns in a table (relation).
- **Syntax:**
- SQL>UPDATE table_name SET column_name = value [, column_name = value ...] [WHERE condition];
- **For example:**
- SQL>UPDATE ucpes SET Author="webmaster" WHERE Author="anonymous";
- **DELETE/FROM/WHERE**
- This command is used for removing one or more rows from a table (relation).
- **Syntax:**
- SQL>DELETE FROM table_name [WHERE condition];
- SQL>DELETE FROM ucpes WHERE Author="unknown";

- **SELECT/FROM/WHERE**

In Oracle, tables are consists of columns and rows. For example, the customers table in the sample database has the following columns: customer_id, name, address, website and credit_limit. The customers table also has data in these columns.

To retrieve data from one or more columns of a table, we use the SELECT statement with the following syntax:

```
SQL>SELECT
  column_1,
  column_2,
  ...
FROM
  table_name;
```

In this SELECT statement:

- First, specify the table name from which we want to query the data.
- Second, indicate the columns from which we want to return the data. If we have more than one column, we need to separate each by a comma (,).

Oracle SELECT examples

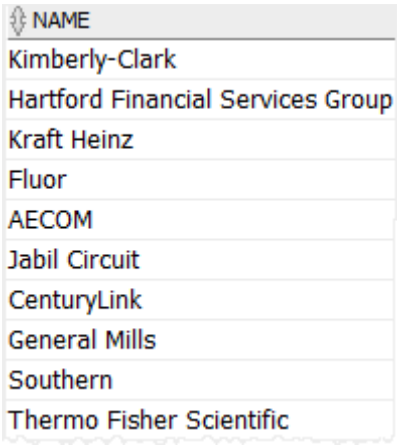
Let’s take some examples of using the Oracle SELECT statement to understand how it works.

A) query data from a single column

To get the customer names from the customers table, we use the following statement:

```
SQL>SELECT
  name
FROM
customers;
```

The following picture illustrates the result:



NAME
Kimberly-Clark
Hartford Financial Services Group
Kraft Heinz
Fluor
AECOM
Jabil Circuit
CenturyLink
General Mills
Southern
Thermo Fisher Scientific

B)Querying data from multiple columns

To query data from multiple columns, we specify a list of comma-separated column names.

The following example shows how to query data from the customer_id, name, and credit_limit columns of the customer table.

```
SQL>SELECT
   customer_id,
   name,
   credit_limit
   FROM
   customers;
```

The following shows the result:

CUSTOMER_ID	NAME	CREDIT_LIMIT
35	Kimberly-Clark	400
36	Hartford Financial Services Group	400
38	Kraft Heinz	500
40	Fluor	500
41	AECOM	500
44	Jabil Circuit	500
45	CenturyLink	500
47	General Mills	600
48	Southern	600
50	Thermo Fisher Scientific	700

C)Querying data from all columns of a table

The following example retrieves all rows from all columns of the customers table:

```
SQL>SELECT
    customer_id,
    name,
    address,
    website,
    credit_limit
    FROM
    customers;
```

Here is the result:

CUSTOMER_ID	NAME	ADDRESS	WEBSITE	CREDIT_LIMIT
1	Raytheon	514 W Superior St, Kokomo, IN	http://www.raytheon.com	100
2	Plains GP Holdings	2515 Bloyd Ave, Indianapolis, IN	http://www.plainsallamerican.com	100
3	US Foods Holding	8768 N State Rd 37, Bloomington, IN	http://www.usfoods.com	100
4	AbbVie	6445 Bay Harbor Ln, Indianapolis, IN	http://www.abbvie.com	100
5	Centene	4019 W 3Rd St, Bloomington, IN	http://www.centene.com	100
6	Community Health Systems	1608 Portage Ave, South Bend, IN	http://www.chs.net	100
7	Alcoa	23943 Us Highway 33, Elkhart, IN	http://www.alcoa.com	100
8	International Paper	136 E Market St # 800, Indianapolis, IN	http://www.internationalpaper.com	100
9	Emerson Electric	1905 College St, South Bend, IN	http://www.emerson.com	100
10	Union Pacific	3512 Rockville Rd # 137C, Indianapolis, IN	http://www.up.com	200

To make it handy, we can use the shorthand asterisk (*) to instruct Oracle to return data from all columns of a table as follows:

```
SQL>SELECT * FROM customers;
```

Oracle Dual Table

In Oracle, the SELECT statement must have a FROM clause. However, some queries don't require any table for example:

```
SQL>SELECT
  UPPER('This is a string')
   FROM
   what_table
```

In this case, we might think about creating a table and use it in the FROM clause for just using the upper() function.

Fortunately, Oracle provides the DUAL table which is a special table that belongs to the schema of the user SYS but is accessible to all users. The DUAL table has one column named DUMMY whose data type is varchar2() and contains one row with a value X.

```
SQL>SELECT   * FROM dual;
```

By using the DUAL table, we can call the upper() function as follows:

```
SQL>SELECT
      UPPER('This is a string')
      FROM
      Dual;
```

Besides calling built-in function, we can use expressions in the SELECT clause of a query that accesses the DUAL table:

```
SQL>SELECT
      (10+ 5)/2
      FROM
     dual;
```

The DUAL table is most simple one because it was designed for fast access.

Oracle ORDER BY Clause

In Oracle, a table stores its rows in unspecified order regardless of the order which rows were inserted into the database. To query rows in either ascending or descending order by a column, we must explicitly instruct Oracle Database that we want to do so.

For example, we may want to list all customers by their names alphabetically or display all customers in order of lowest to highest credit limits.

To sort data, we add the ORDER BY clause to the SELECT statement as follows:

```
SQL>SELECT
   column_1,
   column_2,
   column_3,
   ...
  FROM
   table_name
   ORDER BY
   column_1 [ASC | DESC] [NULLS FIRST | NULLS LAST],
   column_1 [ASC | DESC] [NULLS FIRST | NULLS LAST],
   ...
```

To sort the result set by a column, we list that column after the ORDER BY clause.

Following the column name is a sort order that can be:

- ASC for sorting in ascending order
- DESC for sorting in descending order

By default, the ORDER BY clause sorts rows in ascending order whether we specify ASC or not. If we want to sort rows in descending order, you use DESC explicitly. The ORDER BY clause allows us to sort data by multiple columns where each column may have different sort orders. The ORDER BY clause is always the last clause in a SELECT statement.

Oracle ORDER BY clause examples

We will use the customers table in the sample database for demonstration.

The following statement retrieves customer name, address, and credit limit from the customers table:

```
SQL>SELECT
    name,
    address,
    credit_limit
    FROM
    customers;
```

NAME	ADDRESS	CREDIT_LIMIT
Kimberly-Clark	1660 University Ter, Ann Arbor, MI	400
Hartford Financial Services Group	15713 N East St, Lansing, MI	400
Kraft Heinz	10315 Hickman Rd, Des Moines, IA	500
Fluor	1928 Sherwood Dr, Council Bluffs, IA	500
AECOM	2102 E Kimberly Rd, Davenport, IA	500
Jabil Circuit	221 3Rd Ave Se # 300, Cedar Rapids, IA	500
CenturyLink	2120 Heights Dr, Eau Claire, WI	500
General Mills	6555 W Good Hope Rd, Milwaukee, WI	600
Southern	1314 N Stoughton Rd, Madison, WI	600

A) Sorting rows by a column example

To sort the customer data by names alphabetically in ascending order, we use the following statement:

```
SQL>SELECT
   name,
   address,
   credit_limit
   FROM
   customers
   ORDER BY
   name ASC;
```

NAME	ADDRESS	CREDIT_LIMIT
3M	Via Frenzy 6903, Roma,	1200
ADP	Langstr 14, Zuerich, ZH	700
AECOM	2102 E Kimberly Rd, Davenport, IA	500
AES	33 Fulton St, Poughkeepsie, NY	1200
AIG	12817 Coastal Hwy, Ocean City, MD	2400
AT&T	55 Church Hill Rd, Reading, PA	1200
AbbVie	6445 Bay Harbor Ln, Indianapolis, IN	100
Abbott Laboratories	3310 Dixie Ct, Saginaw, MI	200
Advance Auto Parts	2674 Collingwood St, Detroit, MI	3700
Aetna	200 E Fort Ave, Baltimore, MD	2400

The ASC instructs Oracle to sort the rows in ascending order. Because the ASC is optional. If we omit it, by default, the ORDER BY clause sorts rows by the specified column in ascending order.
Therefore, the following expression:

```
ORDER BY name ASC
```
is equivalent to the following:

```
ORDER BY name
```

To sort customer by name alphabetically in descending order, we explicitly use DESC after the column name in the ORDER BY clause as follows:

```
SQL>SELECT
   name,
   address,
   credit_limit
   FROM
   customers
   ORDER BY
   name DESC;
```

The following picture shows the result that customers sorted by names alphabetically in descending order:

B) Sorting rows by multiple columns example

To sort multiple columns, you separate each column in the ORDER BY clause by a comma.

See the following contacts table in the sample database.

For example, to sort contacts by their first names in ascending order and their last names in descending order, we use the following statement:

```
SQL>SELECT
  first_name,
  last_name
 FROM
  contacts
 ORDER BY
  first_name,
  last_name DESC;
```

In this example, Oracle first sorts the rows by first names in ascending order to make an initial result set. Oracle then sorts the initial result set by the last name in descending order.

See the following result:

Corliss	Mcneil
Cristine	Bell
Daina	Combs
Daniel	Glass
Daniel	Costner
Darron	Robertson
Debra	Herring
Dell	Wilkinson
Delpha	Golden
Deneen	Hays
Denny	Daniel
Diane	Higgins
Dianne	Sen
Dianne	Derek
Dick	Lamb
Don	Hansen
Doretha	Tyler
Dorotha	Wong

In this result:

- First, the first names are sorted in ascending order.
- Second, if two first names are the same, the last names are sorted in descending order e..g, Daniel Glass and Daniel

Costner, Dianne Sen and Dianne Derek, Doretha Tyler and Dorotha Wong.

Oracle SELECT DISTINCT

The DISTINCT clause is used in a SELECT statement to filter duplicate rows in the result set. It ensures that rows returned are unique for the column or columns specified in the SELECT clause.

The following illustrates the syntax of the SELECT DISTINCT statement:

```
SQL>SELECT DISTINCT
      column_1
      FROM
      table;
```

In this statement, the values in the column_1 of the table are compared to determine the duplicates.

To retrieve unique data based on multiple columns, we need to specify the column list in the SELECT clause as follows:

```
SQL>SELECT
   DISTINCT column_1,
   column_2,
      ...
 FROM
   table_name;
```

In this syntax, the combination of values in the column_1, column_2, and column_3 are used to determine the uniqueness of the data.

The DISTINCT clause can be used only in the SELECT statement.

Oracle SELECT DISTINCT examples

Let's look at some examples of using SELECT DISTINCT to see how it works.

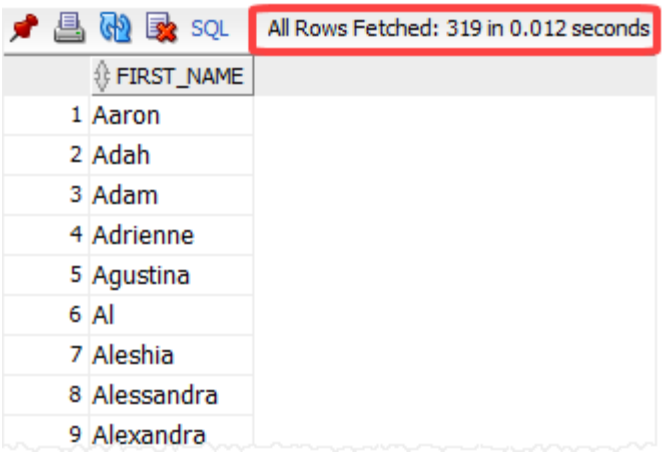
A) Oracle SELECT DISTINCT one column example

See the contacts table in the sample database:

The following example retrieves all contact first names:

```
SQL>SELECT
  first_name
 FROM
  contacts
 ORDER BY
  first_name;
```

The query returned 319 rows, indicating that the contacts table has 319 rows.



To get unique contact first names, we add the DISTINCT keyword to the above SELECT statement as follows:

```
SQL>SELECT DISTINCT
  first_name
  FROM
  contacts
  ORDER BY
  first_name;
```

Now, the result set has 302 rows, meaning that 17 duplicate rows have been removed.

B) Oracle SELECT DISTINCT multiple columns example

See the following order_items table:

The following statement selects distinct product id and quantity from the order_items table:

```
SQL>SELECT
   DISTINCT product_id,
   quantity
   FROM
   ORDER_ITEMS
   ORDER BY
   product_id;
```

The following illustrates the result:

PRODUCT_ID	QUANTITY
1	43
1	57
1	95
1	127
1	135
2	65
2	99
3	46
3	101
4	82

In this example, both values the product_id and quantity columns are used for evaluating the uniqueness of the rows in the result set.

Oracle WHERE Clause

The WHERE clause specifies a search condition for rows returned by the select statement. The following illustrates the syntax of the WHERE clause:

```
SQL>SELECT
   column_1,
   column_2,
   ...
FROM
   table_name
WHERE
   search_condition
ORDER BY
   column_1,
   column_2;
```

The WHERE clause appears after the FROM clause but before the order by clause. Following the WHERE keyword is the search_condition that defines a condition which returned rows must satisfy.

Besides the SELECT statement, WE can use the WHERE clause in the DELETE or UPDATE statement to specify which rows to update or delete.

Oracle WHERE examples

See the following products table in the sample database:

A) Selecting rows by using a simple equality operator

The following example returns only products whose names are 'Kingston':

```
SQL>SELECT
    product_name,
    description,
    list_price,
    category_id
   FROM
    products
   WHERE
    product_name = 'Kingston';
```

The following picture illustrates the result:

PRODUCT_NAME	DESCRIPTION	LIST_PRICE	CATEGORY_ID
Kingston	Speed:DDR3-1333,Type:240-pin DIMM,CAS:9Module:4x16GBSize:64GB	671.38	5
Kingston	Speed:DDR3-1600,Type:240-pin DIMM,CAS:11Module:4x8GBSize:32GB	653.5	5
Kingston	Speed:DDR3-1600,Type:240-pin DIMM,CAS:11Module:4x16GBSize:64GB	644	5
Kingston	Speed:DDR4-2133,Type:288-pin DIMM,CAS:15Module:4x16GBSize:64GB	741.63	5

In this example, Oracle evaluates the clauses in the following order:FROM WHERE and SELECT

1. First, the FROM clause specified the table for querying data.
2. Second, the WHERE clause filtered rows based on the condition e.g., product_name = 'Kingston').
3. Third, the SELECT clause chose the columns that should be returned.

B) Select rows using comparison operator

Besides the equality operator, Oracle provides us with many other comparison operators illustrated in the following table:

Operator	Description
=	Equality
!=,<>	Inequality
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
IN	Equal to any value in a list of values
ANY/ SOME / ALL	Compare a value to a list or subquery. It must be preceded by another operator such as =, >, <.
NOT IN	Not equal to any value in a list of values
[NOT] BETWEEN *n* and *m*	Equivalent to [Not] >= *n* and <= *y*.
[NOT] EXISTS	Return true if subquery returns at least one row
IS [NOT] NULL	NULL test

For example, to get products whose list prices are greater than 500, we use the following statement:

```
SQL>SELECT
    product_name,
    list_price
   FROM
    products
   WHERE
    list_price > 500;
```

PRODUCT_NAME	LIST_PRICE
Gigabyte GA-Z270X-Gaming 9	503.98
Asus Rampage V Edition 10	519.99
Supermicro H8DG6-F	525.99
MSI X99A GODLIKE GAMING CARBON	549.59
Asus Z10PE-D8 WS	561.59
Asus RAMPAGE V EXTREME	572.96
Asus ROG MAXIMUS IX EXTREME	573.99
Asus X99-E-10G WS	649
Intel DP35DPM	789.79

C) Select rows that meet some conditions

To combine conditions you can use the AND, OR and NOT logical operators.

For example, to get all motherboards that belong to the category id 1 and have list prices greater than 500, we use the following statement:

```
SQL>SELECT
    product_name,
    list_price
   FROM
    products
   WHERE
    list_price > 500
    AND category_id = 4;
```

The result set includes only motherboards whose list prices are greater than 500.

PRODUCT_NAME	LIST_PRICE
Gigabyte GA-Z270X-Gaming 9	503.98
Asus Rampage V Edition 10	519.99
Supermicro H8DG6-F	525.99
MSI X99A GODLIKE GAMING CARBON	549.59
Intel Core i7-5930K	554.99
Asus Z10PE-D8 WS	561.59
Intel Xeon E5-1650 V3	564.89
Asus RAMPAGE V EXTREME	572.96
Asus ROG MAXIMUS IX EXTREME	573.99

D) Selecting rows that have a value between two values

To find rows that have a value between two values, we use the BETWEEN operator in the WHERE clause.

For example, to get the products whose list prices are between 650 and 680, we use the following statement:

```
SQL>SELECT
    product_name,
    list_price
  FROM
 products
 WHERE
   list_price BETWEEN 650 AND 680
 ORDER BY
   list_price;
```

The following picture illustrates the result set:

PRODUCT_NAME	LIST_PRICE
Kingston	653.5
Corsair Dominator Platinum	659.99
Intel Core i7-3930K	660
Kingston	671.38
G.Skill Ripjaws V Series	677.99
Intel Core i7-7820X	678.75

Note that the following expressions are equivalent:

list_price BETWEEN 650 AND 680
list_price >= 650 AND list_price <= 680

E) Selecting rows that are in a list of values

To query rows that are in a list of values, we use the IN operator as follows:

```
SQL>SELECT
    product_name,
    category_id
   FROM
    products
   WHERE
    category_id IN(1, 4)
   ORDER BY
    product_name;
```

The following illustrates the result:

PRODUCT_NAME	CATEGORY_ID
AMD Opteron 6378	1
ASRock C2750D4I	4
ASRock E3C224D4M-16RE	4
ASRock EP2C602-4L/D16	4
ASRock EP2C612 WS	4
ASRock Fatal1ty X299 Professional Gaming i9	4
ASRock X299 Taichi	4
ASRock X99 Extreme11	4
ASRock Z270 SuperCarrier	4
Asus KGPE-D16	4

The expression:

category_id IN (1, 4)
is the same as:

category_id = 1 OR category_id = 4

F) Selecting rows which contain value as a part of a string

The following statement retrieves product whose name starts with Asus:

```
SQL>SELECT
  product_name,
  list_price
 FROM
  products
 WHERE
  product_name LIKE 'Asus%'
 ORDER BY
  list_price;
```

In this example, we used the LIKE operator to match rows based on the specified pattern.

Oracle Alias

When we query data from a table, Oracle uses the column names of the table for displaying the column heading. For example, the following statement returns the first name and last name of employees:

```
SQL>SELECT
  first_name,
  last_name
  FROM
  employees
  ORDER BY
  first_name;
```

FIRST_NAME	LAST_NAME
Aaron	Patterson
Abigail	Palmer
Albert	Watson
Alex	Sanders
Alice	Wells
Amber	Rose
Amelia	Myers
Amelie	Hudson
Annabelle	Dunn
Austin	Flores

In this example, first_name and last_name column names were quite clear. However, sometimes, the column names are quite vague for describing the meaning of data such as:

```
SQL>SELECT
  lstprc,
  prdnm
  FROM
  long_table_name;
```

To better describe the data displayed in the output, we can substitute a column alias for the column name in the query results.

For instance, instead of using first_name and last_name, we might want to use forename and surname for display names of employees.

To instruct Oracle to use a column alias, we simply list the column alias next to the column name in the SELECT clause as shown below:

```
SQL>SELECT
  first_name AS forename,
  last_name  AS surname
```

```
FROM
 employees;
```

FORENAME	SURNAME
Summer	Payne
Rose	Stephens
Annabelle	Dunn
Tommy	Bailey
Blake	Cooper
Jude	Rivera
Tyler	Ramirez
Ryan	Gray
Elliot	Brooks
Elliott	James

The AS keyword is used to distinguish between the column name and the column alias. Because the AS keyword is optional, we can skip it as follows:

```
SELECT
  first_name forename,
  last_name surname
FROM
  employees;
```

Using Oracle column alias to make column heading more meaningful.

By default, Oracle capitalizes the column heading in the query result. If we want to change the letter case of the column heading, we need to enclose it in quotation marks (“”).

```
SQL> SELECT
  first_name "Forename",
  last_name "Surname"
 FROM
  employees;
```

Forename	Surname
Summer	Payne
Rose	Stephens
Annabelle	Dunn
Tommy	Bailey
Blake	Cooper
Jude	Rivera
Tyler	Ramirez
Ryan	Gray
Elliot	Brooks

As shown in the output, the forename and surname column headings retain their letter cases.

STUDENTS' LABORATORY ACTIVITY

1. Create the EMP table.
2. Show the Structure of EMP table.
3. Create the DEPT table.
4. Show the Structure of DEPT table.
5. Insert 5 rows into the DEPT table.
6. Insert 5 rows into the EMP table.
7. Display all data from EMP table.
8. Display all data from DEPT table.
9. Display unique jobs from the EMP table.
10. Write a query to Name the column headings EMP#, Employee, Job and Hire date respectively.
11. Create a query to display the Name and salary of employees earning more than Rs.2850. Save the query and run it.
12. Display the employee name, job and start date of employees hire date between Feb.20.1981 and May 1, 1981. Order the query in ascending order of start date.
13. Display the name and title of all employees who don't have a Manager.
14. Display the name, salary and comm. For all employee who earn comm. Sort data in descending order of salary and comm.
15. Write a query to display the date. Label the column DATE.

16. Delete the information of student having roll No -15 and City- Bhubaneswar. Rename the Student database table to STUDENT INFORMATION.

SQL Functions,Set Operators ,Joins and Sub Queries

SQL NUMERIC FUNCTIONS

Function	Action	Example	Displays
CEIL(*n*)	Returns nearest whole number greater than or equal to *n*.	select CEIL(12.3) from dual;	13
FLOOR(*n*)	Returns nearest whole number less than or equal to *n*.	select FLOOR(127.6) from dual;	127
ROUND(*n*,*m*)	Rounds *n* to *m* places to the right of the decimal point.	select ROUND(579.34886,3) from dual;	579.349
POWER(*m*,*n*)	Multiplies *m* to the power *n*.	select POWER(5,3) from dual;	125

MOD(*m*,*n*)	Returns the remainder of the division of *m* by *n*. If *n* = 0, then 0 is returned. If *n* > *m*, then *m* is returned.	select MOD(9,5) from dual;
		select mod(10,5) from dual;
		select mod(6,7) from dual;
			4
			0
			6

SQRT(*n*)	Returns the square root of *n*.	select SQRT(9) from dual;	3
ABS(n)	Returns the absolute value of *n*.	select ABS(-29) from dual;	29
TRUNC (n1, n2)	Returns a value with the required number of decimal places while a negative *n2* rounds to the left of the decimal.	select TRUNC(29.16, 1), trunc(31.2,-1) from dual;	29.1 30

SQL CHARACTER FUNCTIONS

Function	Action	Example	Displays
LOWER(*char*)	Converts the entire string to lowercase.	select LOWER('DAliA') from dual;	dalia
REPLACE(*char, str1,str2*)	Replaces every occurrence of *str1* in *char* with *str2*.	select REPLACE('Scott', 'S', 'Boy') from dual;	Boycott
SUBSTR(*char, m,n*)	Extracts the characters from *char* starting in position *m* for *n* characters.	select SUBSTR('ABCDEF',4,2) from dual;	DE
TRIM(*char*)	Removes spaces before and after *char*.	Select TRIM(' testing ') from dual;	testing
LENGTH(*char*)	Returns the length of *char*.	select LENGTH('Marissa') from dual;	7
RPAD(*expr1, n,expr2*)	Pads *expr1* with *expr2* to the right for *n* characters. Often used for space padding in the creation of a fixed-length record.	select RPAD('Amanda', 10, '1') from dual;	Amanda1111
INITCAP(*char*)	Changes the first character of each element in *char* to uppercase.	select INITCAP('shane k.') from dual;	Shane K.

SQL AGGREGATE FUNCTIONS

Function	Action	Example	Displays
COUNT(*expr*)	Returns a count of non-null column values for each row retrieved	select COUNT(cust_id) from customers where cust_state_ province = 'NY';	694
AVG(*expr*)	Returns the average for the column values and rows selected	select AVG(amount_sold) from sales where prod_id = 117;	9.92712978
SUM(*expr*)	Returns the sum of the column values for all the retrieved rows	select SUM(amount_sold) from sales where prod_id = 117;	170270.13
MIN(*expr*)	Returns the minimum value for the column and rows retrieved	select MIN(prod_list_price) from products;	6.99
MAX(*expr*)	Returns the maximum value for the column and rows retrieved	select MAX(prod_list_price) from products;	1299.99

Oracle AND Operator

The AND operator is a logical operator that combines Boolean expressions and returns true if both expressions are true. If one of the expressions is false, the AND operator returns false.
The syntax of the AND operator is as follows:

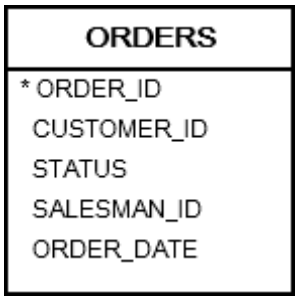
expression_1 AND expression_2

Typically, AND is used in the WHERE clause of the SELECT, DELETE, and UPDATE statements to form a condition for matching data. In addition, we use the AND operator in the predicate of the JOIN clause to form the join condition.

When we use more than one logical operator in a statement, Oracle always evaluates the AND operators first. However, we can use parentheses to change the order of evaluation.

Oracle AND operator examples

See the following orders table in the sample database:



A) Oracle AND to combine two Boolean expressions example

The following example finds orders of the customer 2 with the pending status:

```
SQL>SELECT
```

```
  order_id,
  customer_id,
  status,
  order_date
 FROM
  orders
 WHERE
  status = 'Pending'
AND customer_id = 2
ORDER BY
order_date;
```

In this example, the query returned all orders that satisfy both expressions:

status = 'Pending'
and

customer_id = 2
Here is the result:

ORDER_ID	CUSTOMER_ID	STATUS	ORDER_DATE
78	2	Pending	14-DEC-15
44	2	Pending	20-FEB-17

B) Oracle AND to combine more than two Boolean expressions example

we can use multiple AND operators to combine Boolean expressions.
For example, the following statement retrieves the orders that meet all the following conditions:

- placed in 2017
- is in charge of the salesman id 60

- has the shipped status.

- SQL>SELECT
- order_id,
- customer_id,
- status,
- order_date
- FROM
- orders
- WHERE
- status = 'Shipped'
- AND salesman_id = 60
- AND EXTRACT(YEAR FROM order_date) = 2017
- ORDER BY
- order_date;

ORDER_ID	CUSTOMER_ID	STATUS	ORDER_DATE
77	1	Shipped	02-JAN-17
99	49	Shipped	07-JAN-17
104	18	Shipped	01-FEB-17

- in this example, we used the EXTRACT() function to get the YEAR field from the order date and compare it with 2017.

- **C) Oracle AND to combine with OR operator example**

- we can combine the AND operator with other logical operators such as OR and NOT to form a condition.
- For example, the following query finds order placed by customer id 44 and has status canceled or pending.

SQL>SELECT order_id, customer_id, status, salesman_id, order_date FROM orders WHERE (status = 'Canceled' OR status = 'Pending') AND customer_id = 44 ORDER BY order_date;

ORDER_ID	CUSTOMER_ID	STATUS	SALESMAN_ID	ORDER_DATE
10	44	Pending	(null)	24-JAN-17
69	44	Canceled	54	17-MAR-17

Oracle OR Operator

The OR operator is a logical operator that combines Boolean expressions and returns true if one of the expressions is true.

The following illustrates the syntax of the OR operator:

expression_1 AND expression_2

We often use the OR operator in the WHERE clause of the SELECT, DELETE, and UPDATE statements to form a condition for filtering data.

If we use multiple logical operators in a statement, Oracle evaluates the OR operators after the NOT and AND operators. However, we can change the order of evaluation by using parentheses.

Oracle OR operator examples

We will use the orders table in the sample database for the demonstration.

A)using Oracle OR operator to combine two Boolean expressions example

The following example finds orders whose status is pending or canceled:

```
SQL>SELECT
    order_id,
    customer_id,
    status,
    order_date
   FROM
    orders
   WHERE
    status = 'Pending'
    OR status = 'Canceled'
  ORDER BY
    order_date DESC;
```

In this example, the statement returned all orders that satisfy one of the following expressions:

status = 'Pending'
status = 'Canceled'

The following picture illustrates the result:

ORDER_ID	CUSTOMER_ID	STATUS	ORDER_DATE
1	4	Pending	15-OCT-17
28	6	Canceled	15-AUG-17
31	46	Canceled	12-AUG-17
21	21	Pending	27-MAY-17
5	5	Canceled	09-APR-17
69	44	Canceled	17-MAR-17
70	45	Canceled	21-FEB-17
44	2	Pending	20-FEB-17
46	58	Pending	20-FEB-17
10	44	Pending	24-JAN-17

B) Using Oracle OR operator to combine more than two Boolean expressions example

We often use the OR operators to combine more than two Boolean expressions. For example, the following statement

retrieves the orders which are in charge of one of the following the salesman id 60, 61 or 62:

```
SQL>SELECT
    order_id,
    customer_id,
    status,
    salesman_id,
    order_date
   FROM
    orders
   WHERE
    salesman_id = 60
    OR salesman_id = 61
    OR salesman_id = 62
    ORDER BY
    order_date DESC;
```

Here is the result:

ORDER_ID	CUSTOMER_ID	STATUS	SALESMAN_ID	ORDER_DATE
88	6	Shipped	61	01-NOV-17
94	1	Shipped	62	27-OCT-17
60	1	Shipped	62	30-JUN-17
40	55	Shipped	62	11-MAY-17
70	45	Canceled	61	21-FEB-17
46	58	Pending	62	20-FEB-17
104	18	Shipped	60	01-FEB-17
99	49	Shipped	60	07-JAN-17
77	1	Shipped	60	02-JAN-17
102	45	Shipped	61	20-DEC-16

Instead of using multiple OR operators, we can use the IN operator as shown in the following example:

```
SQL>SELECT
    order_id,
    customer_id,
```

```
    status,
    salesman_id,
    order_date
   FROM
    orders
   WHERE
    salesman_id IN(
        60,
        61,
        62
         )
  ORDER BY
    order_date DESC;
```

This query returns the same result as the one that uses the OR operator above.

C) Using Oracle OR operator to combine with AND operator example

We can combine the OR operator with other logical operators such as AND and NOT to form a condition. For example, the following query returns the orders that belong to the customer id 44 and have canceled or pending status.

```
SQL>SELECT
    order_id,
    customer_id,
    status,
    salesman_id,
    order_date
    FROM
    orders
    WHERE
    (
        status = 'Canceled'
        OR status = 'Pending'
```

```
    )
    AND customer_id = 44
  ORDER BY
    order_date;
```

ORDER_ID	CUSTOMER_ID	STATUS	SALESMAN_ID	ORDER_DATE
10	44	Pending	(null)	24-JAN-17
69	44	Canceled	54	17-MAR-17

Oracle NOT IN example

The example shows how to find orders whose statuses are not Shipped and Canceled:

```
SQL>SELECT
    order_id,
    customer_id,
    status,
    salesman_id
   FROM
    orders
  WHERE
    status NOT IN(
        'Shipped',
        'Canceled'
    )
ORDER BY
order_id;
```

The result is:

ORDER_ID	CUSTOMER_ID	STATUS	SALESMAN_ID
1	4	Pending	56
10	44	Pending	(null)
16	16	Pending	(null)
21	21	Pending	(null)
44	2	Pending	55
46	58	Pending	62
50	62	Pending	55
55	66	Pending	59
68	9	Pending	(null)

Oracle IN subquery example

The following example returns the id, first name, and last name of salesmen who are in charge of orders that were canceled

```
SQL>SELECT
    employee_id,
    first_name,
    last_name
   FROM
    employees
   WHERE
    employee_id IN(
        SELECT
            DISTINCT salesman_id
           FROM
            orders
          WHERE
            status = 'Canceled'
            )
        ORDER BY
        first_Name;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
61	Daisy	Ortiz
56	Evie	Harrison
64	Florence	Freeman
62	Freya	Gomez
55	Grace	Ellis
60	Isabelle	Marshall
54	Lily	Fisher
57	Scarlett	Gibson

In this example, the subquery executes first and returns a list of salesman ids:

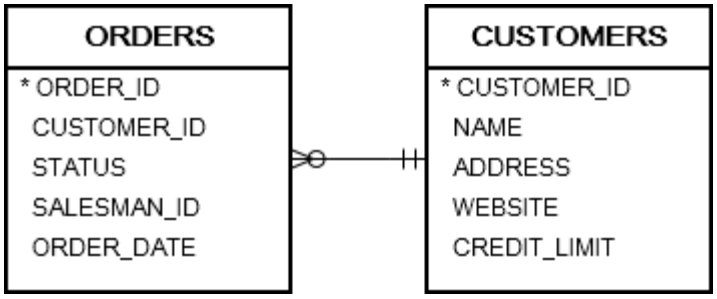
```
SQL>SELECT
    DISTINCT salesman_id
    FROM
    orders
    WHERE
    status = 'Canceled'
```

SALESMAN_ID
57
(null)
54
55
61
56
64
62
60

And these salesman ids are used for the outer query which finds all employees whose ids are equal to any id in the salesman id list

Oracle NOT IN subquery example

See the following customers and orders tables:



The following example uses the NOT IN to find customers who have not placed any orders:

```
SQL>SELECT
   customer_id,
   name
   FROM
   customers
   WHERE
   customer_id NOT IN(
      SELECT
         customer_id
      FROM
     orders
      );
```

CUSTOMER_ID	NAME
35	Kimberly-Clark
36	Hartford Financial Services Group
38	Kraft Heinz
40	Fluor
72	Icahn Enterprises
74	Performance Food Group
76	DISH Network
77	FirstEnergy
80	AES
81	CarMax

E) Oracle IN vs. OR

The following example shows how to get the sales orders of salesman 60, 61, and 62:

```
SQL>SELECT
   customer_id,
   status,
   salesman_id
   FROM
   orders
  WHERE
   salesman_id IN(
      60,
      61,
      62
     )
   ORDER BY
     customer_id;
```

CUSTOMER_ID	STATUS	SALESMAN_ID
1	Shipped	62
1	Shipped	62
1	Shipped	60
3	Shipped	62
4	Shipped	61
5	Pending	60
6	Shipped	61
7	Canceled	61
8	Canceled	61
16	Shipped	62

It is equivalent to:

```
SQL>SELECT
   customer_id,
   status,
   salesman_id
   FROM
   orders
  WHERE
   salesman_id = 60
   OR salesman_id = 61
   OR salesman_id = 62
  ORDER BY
```

customer_id;
Note that the expression:

salesman_id NOT IN (60,61,62);
has the same effect as:

salesman_id != 60
AND salesman_id != 61
AND salesman_id != 62;

Oracle LIKE

Sometimes, you want to query data based on a specified pattern. For example, you may want to find contacts whose last names start with 'St' or first names end with 'er'. In this case, we use the Oracle LIKE operator.

The syntax of the Oracle LIKE operator is as follows:

expresion [NOT] LIKE pattern
In this syntax, we have:

1) expression

The expression is a column name or an expression that we want to test against the pattern.

2) pattern

The pattern is a string to search for in the expression. The pattern includes the following wildcard characters:

- % (percent) matches any string of zero or more character.
- _ (underscore) matches any single character.

The LIKE operator returns true if the expression matches the pattern. Otherwise, it returns false.

The NOT operator, if specified, negates the result of the LIKE operator.

Oracle LIKE examples

Let’s take some examples of using the Oracle LIKE operator to see how it works.

We will use the contacts table in the sample database for the demonstration:

A) % wildcard character examples

The following example uses the % wildcard to find the phones of contacts whose last names start with 'St':

```
SQL>SELECT
    first_name,
    last_name,
    phone
   FROM
    contacts
  WHERE
    last_name LIKE 'St%'
```

```
ORDER BY
 last_name;
```

The following picture illustrates the result:

FIRST_NAME	LAST_NAME	PHONE
Josie	Steele	+41 69 012 3581
Bill	Stein	+39 6 012 4501
Birgit	Stephenson	+1 608 123 4374
Herman	Stokes	+39 49 012 4777
Violeta	Stokes	+1 810 123 4212
Gonzalo	Stone	+1 301 123 4814
Flor	Stone	+1 317 123 4104

In this example, we used the pattern:

'St%'
The LIKE operator matched any string that starts with 'St' and is followed by any

number of characters e.g., Stokes, Stein, or Steele, etc.

To find the phone numbers of contacts whose last names end with the string 'er', you use the following statement:

```
SQL>SELECT
    first_name,
    last_name,
    phone
   FROM
    contacts
  WHERE
    last_name LIKE '%er'
  ORDER BY
    last_name;
```

Here is the result:

FIRST_NAME	LAST_NAME	PHONE
Shamika	Bauer	+91 11 012 4853
Stephaine	Booker	+39 55 012 4559
Charlene	Booker	+41 61 012 3537
Annice	Boyer	+1 518 123 4618
Shelia	Brewer	+49 89 012 4129
Annabelle	Butler	+91 80 012 3737
Nichol	Carter	+91 11 012 4813
Barbie	Carter	+41 5 012 3573
Sharee	Carver	+1 215 123 4738
Agustina	Conner	+1 612 123 4399
Daniel	Costner	+1 812 123 4153

The pattern:

%er
matches any string that ends with the 'er' string.

To perform a case-insensitive match, we use either LOWER() or UPPER() function as follows:

UPPER(last_name) LIKE 'ST%'

LOWER(last_name LIKE 'st%'

For example, the following statement finds emails of contacts whose first names start with CH:

```
SQL>SELECT
    first_name,
last_name,
    email
FROM
    contacts
WHERE
    UPPER( first_name ) LIKE 'CH%';
ORDER BY
    first_name;
```

Here is the result:

FIRST_NAME	LAST_NAME	EMAIL
Charlene	Booker	charlene.booker@republicservices.com
Charlie	Sutherland	charlie.sutherland@up.com
Charlie	Pacino	charlie.pacino@amgen.com
Charlsie	Lindsey	charlsie.lindsey@berkshirehathaway.com
Charlsie	Carey	charlsie.carey@group1auto.com
Christal	Grant	christal.grant@gs.com
Christian	Cage	christian.cage@emerson.com

The following example uses the NOT LIKE operator to find contacts whose phone numbers do not start with '+1':

```
SQL>SELECT
  first_name, last_name, phone
FROM
  contacts
WHERE
  phone NOT LIKE '+1%'
ORDER BY
  first_name;
```

The result is:

FIRST_NAME	LAST_NAME	PHONE
Adah	Myers	+41 3 012 3553
Adam	Jacobs	+91 80 012 3699
Adrienne	Lang	+39 2 012 4771
Aleshia	Reese	+41 4 012 3563
Alessandra	Estrada	+41 56 012 3527
Amber	Brady	+91 80 012 3837
Annabelle	Butler	+91 80 012 3737
Annelle	Lawrence	+39 10 012 4379
Arlette	Thornton	+91 80 012 3719

B) _ wildcard character examples

The following example finds the phone numbers and emails of contacts whose first names have the following pattern 'Je_i':

```
SQL>SELECT
```

```
    first_name,
    last_name,
    email,
    phone
   FROM
    contacts
   WHERE
    first_name LIKE 'Je_i'
   ORDER BY
    first_name;
```

Here is the result:

FIRST_NAME	LAST_NAME	EMAIL	PHONE
Jeni	Levy	jeni.levy@centene.com	+1 812 123 4129
Jeri	Randall	jeri.randall@nike.com	+49 90 012 4131

The pattern 'Je_i' matches any string that starts with 'Je', followed by one character, and then followed by 'i' e.g., Jeri or Jeni, but not Jenni.

C) Mixed wildcard characters example

We can mix the wildcard characters in a pattern. For example, the following statement finds contacts whose first names start with Je followed by two characters and then any number of characters. In other words, it will match any last name that starts with Je and has at least 3 characters:

```
SQL>SELECT
    first_name,
    last_name,
    email,
    phone
   FROM
    contacts
```

```
WHERE
 first_name LIKE 'Je_%';
```

FIRST_NAME	LAST_NAME	EMAIL	PHONE
Jeannie	Poole	jeannie.poole@aboutmcdonalds.com	+91 80 012 4637
Jeni	Levy	jeni.levy@centene.com	+1 812 123 4129
Jeri	Randall	jeri.randall@nike.com	+49 90 012 4131
Jerica	Brooks	jerica.brooks@northropgrumman.com	+91 11 012 4811
Jermaine	Cote	jermaine.cote@wfscorp.com	+49 91 012 4133
Jess	Nguyen	jess.nguyen@searsholdings.com	+39 2 012 4773
Jessika	Merritt	jessika.merritt@bnymellon.com	+1 612 123 4397

Oracle Joins

Oracle join is used to combine columns from two or more tables based on values of the related columns. The related columns are typically the primary key column(s) of the first table and foreign key column(s) of the second table.

Oracle supports inner join, left join, right join, full outer join and cross join.

Note that you can join a table to itself to query hierarchical data using an inner join, left join, or right join. This kind of join is known as self-join.

Setting up sample tables

We will create two new tables with the same structure for the demonstration:

```
SQL>CREATE TABLE palette_a (
   id INT PRIMARY KEY,
   color VARCHAR2 (100) NOT NULL
```

```
);

SQL>CREATE TABLE palette_b (
   id INT PRIMARY KEY,
   color VARCHAR2 (100) NOT NULL
);

SQL>INSERT INTO palette_a (id, color)
VALUES (1, 'Red');

SQL>INSERT INTO palette_a (id, color)
VALUES (2, 'Green');
SQL>INSERT INTO palette_a (id, color)
VALUES (3, 'Blue');

SQL>INSERT INTO palette_a (id, color)
VALUES (4, 'Purple');

-- insert data for the palette_b
SQL>INSERT INTO palette_b (id, color)
VALUES (1, 'Green');

SQL>INSERT INTO palette_b (id, color)
VALUES (2, 'Red');
SQL>INSERT INTO palette_b (id, color)
VALUES (3, 'Cyan');

SQL>INSERT INTO palette_b (id, color)
VALUES (4, 'Brown');
```

The tables have some common colors such as Red and Green. Let's call the palette_a the left table and palette_b the right table:

ID	COLOR
1	Red
2	Green
3	Blue
4	Purple

ID	COLOR
1	Green
2	Red
3	Cyan
4	Brown

Oracle inner join

The following statement joins the left table to the right table using the values in the color column:

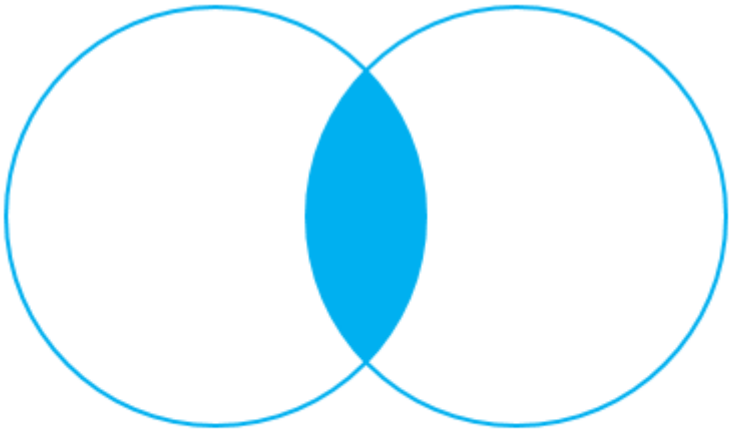
```
SQL>SELECT
   a.id id_a,
   a.color color_a,
   b.id id_b,
   b.color color_b
FROM
   palette_a a
INNER JOIN palette_b b ON a.color = b.color;
```

Here is the output:

ID_A	COLOR_A	ID_B	COLOR_B
2	Green	1	Green
1	Red	2	Red

As can be seen clearly from the result, the inner join returns rows from the left table that match with the rows from the right table.

The following Venn diagram illustrates an inner join when combining two result sets:



INNER JOIN

Oracle left join

The following statement joins the left table with the right table using a left join (or a left outer join):

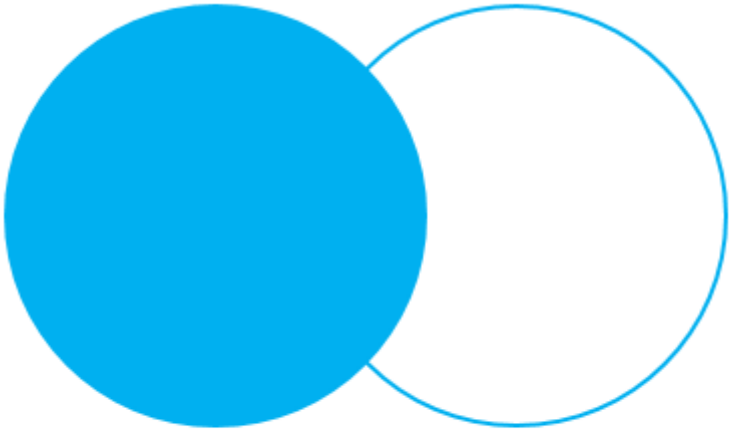
```
SQL>SELECT
    a.id id_a,
    a.color color_a,
    b.id id_b,
    b.color color_b
FROM
    palette_a a
LEFT JOIN palette_b b ON a.color = b.color;
```

The output is shown as follows:

ID_A	COLOR_A	ID_B	COLOR_B
2	Green	1	Green
1	Red	2	Red
3	Blue	(null)	(null)
4	Purple	(null)	(null)

The left join returns all rows from the left table with the matching rows if available from the right table. If there is no matching row found from the right table, the left join will have null values for the columns of the right table:

The following Venn diagram illustrates the left join:



LEFT OUTER JOIN

Sometimes, we want to get only rows from the left table that do not exist in the right table. To achieve this, we use the left join and a WHERE clause to exclude the rows from the right table.

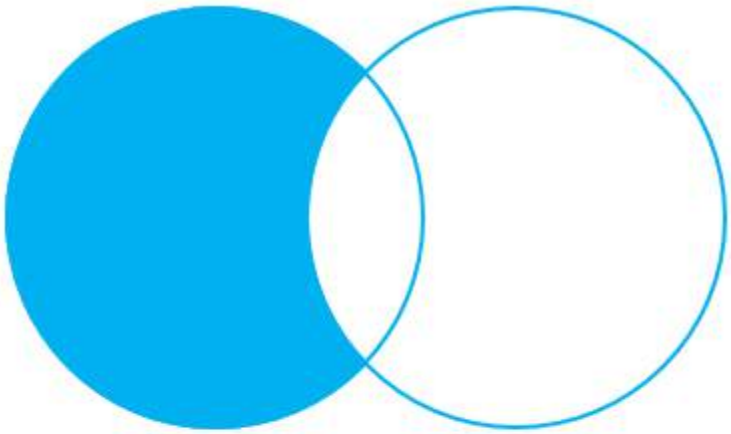
For example, the following statement shows colors that only available in the palette_a but not palette_b:

```
SQL>SELECT
    a.id id_a,
    a.color color_a,
    b.id id_b,
    b.color color_b
    FROM
    palette_a a
    LEFT JOIN palette_b b ON a.color = b.color
   WHERE b.id IS NULL;
```

Here is the output:

ID_A	COLOR_A	ID_B	COLOR_B
3	Blue	(null)	(null)
4	Purple	(null)	(null)

The following Venn diagram illustrates the left join with the exclusion of rows from the right table:



LEFT OUTER JOIN – only rows from the left table

Oracle right join

The right join or right outer join is a reversed version of the left join. The right join makes a result set that contains all rows from the right table with the matching rows from the left table. If there is no match, the left side will have nulls.

The following example use right join to join the left table to the right table:

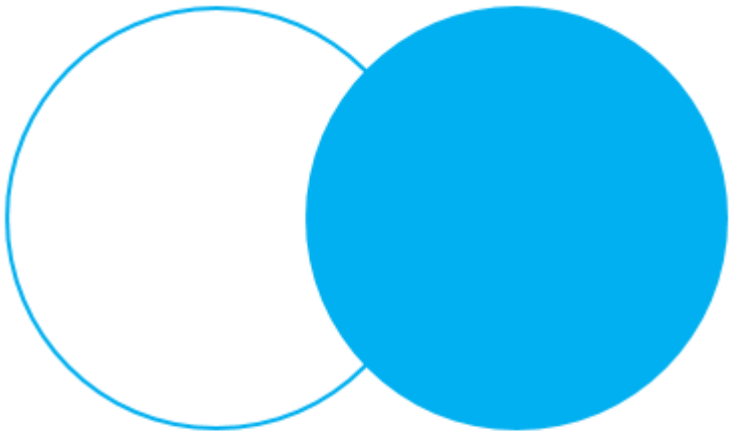
```
SQL>SELECT
    a.id id_a,
    a.color color_a,
    b.id id_b,
    b.color color_b
    FROM
```

```
    palette_a a
  RIGHT JOIN palette_b b ON a.color = b.color;
```

Here is the output:

ID_A	COLOR_A	ID_B	COLOR_B
1	Red	2	Red
2	Green	1	Green
(null)	(null)	4	Brown
(null)	(null)	3	Cyan

The following Venn diagram illustrates the right join:



Likewise, we can get only rows from the right table but not the left table by adding a WHERE clause to the above statement as shown in the following query:

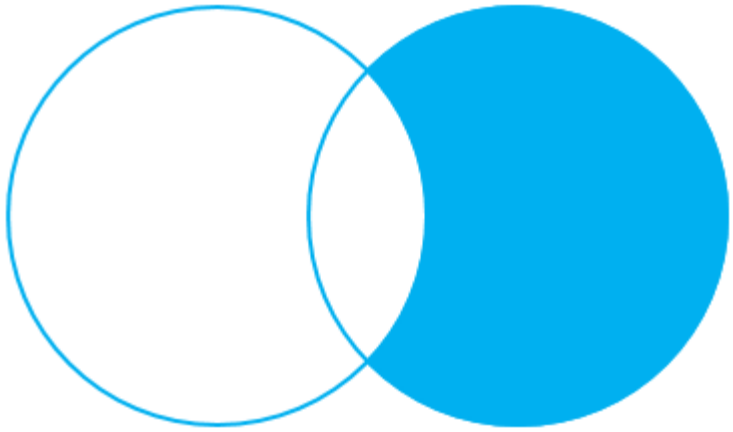
```
SQL>SELECT
    a.id id_a,
    a.color color_a,
    b.id id_b,
    b.color color_b
    FROM
    palette_a a
    RIGHT JOIN palette_b b ON a.color = b.color
```

WHERE a.id IS NULL;

Here is the output:

ID_A	COLOR_A	ID_B	COLOR_B
(null)	(null)	4	Brown
(null)	(null)	3	Cyan

The following Venn diagram illustrates the right join with the exclusion of rows from the left table:



RIGHT OUTER JOIN – only rows from the right table

Oracle full outer join

Oracle full outer join or full join returns a result set that contains all rows from both left and right tables, with the matching rows from both sides where available. If there is no match, the missing side will have nulls.

The following example shows the full outer join of the left and right tables:

SQL>SELECT

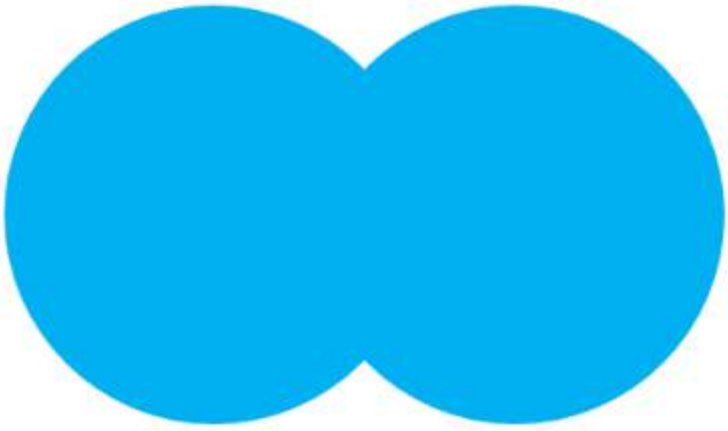
```
    a.id id_a,
    a.color color_a,
    b.id id_b,
    b.color color_b
    FROM
    palette_a a
    FULL OUTER JOIN palette_b b ON a.color = b.color;
```

The following picture illustrates the result set of the full outer join:

ID_A	COLOR_A	ID_B	COLOR_B
1	Red	2	Red
2	Green	1	Green
3	Blue	(null)	(null)
4	Purple	(null)	(null)
(null)	(null)	4	Brown
(null)	(null)	3	Cyan

Note that the OUTER keyword is optional.

The following Venn diagram illustrates the full outer join:



FULL OUTER JOIN

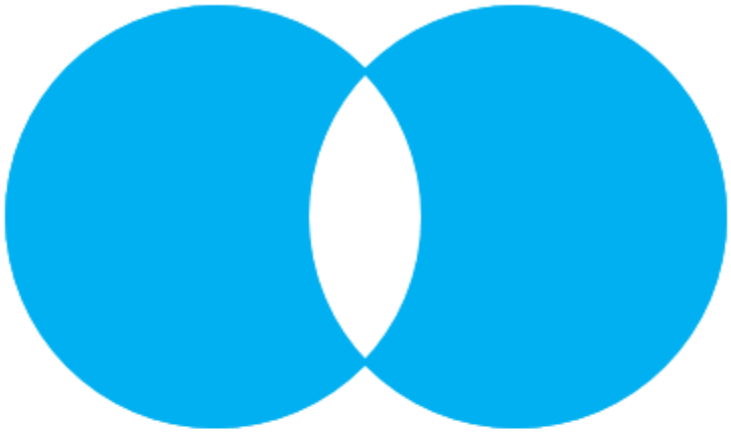
To get a set of rows that are unique from the left and right tales, you perform the same full join and then exclude the rows that you don’t want from both sides using a WHERE clause as follows:

```
SQL>SELECT
    a.id id_a,
    a.color color_a,
    b.id id_b,
    b.color color_b
    FROM
    palette_a a
    FULL JOIN palette_b b ON a.color = b.color
    WHERE a.id IS NULL OR b.id IS NULL;
```

Here is the result:

ID_A	COLOR_A	ID_B	COLOR_B
(null)	(null)	3	Cyan
(null)	(null)	4	Brown
3	Blue	(null)	(null)
4	Purple	(null)	(null)

The following Venn diagram illustrates the above operation:



FULL OUTER JOIN – only
rows unique to both tables

A self join is a join that joins a table with itself. A self join is useful for comparing rows within a table or querying hierarchical data.

A self join uses other joins such as inner join and left join. In addition, it uses the table alias to assign the table different names in the same query.

Note that referencing the same table more than once in a query without using table aliases cause an error.

The following illustrates how the table T is joined with itself:

```
SQL>SELECT
   column_list
   FROM
   T t1
   INNER JOIN T t2 ON
   join_predicate;
```

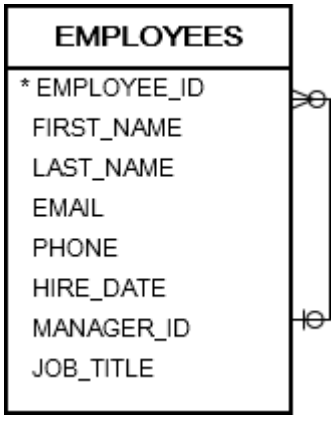
Note that besides the inner join, you can use the left join in the above statement.

Oracle Self Join example

Let's look at some examples of using Oracle self join.

A) Using Oracle self join to query hierarchical data example

See the following employees table in the sample database.



The employees table stores personal information such as id, name, job title. In addition, it has the manager_id column that stores the reporting lines between employees.

The President of the company, who does not report to anyone, has a NULL value in the manager_id column. Other employees, who have a manager, have a numeric value in the manager_id column, which indicates the id of the manager.

To retrieve the employee and manager data from the employees table, we use a self join as shown in the following statement:

```
SQL>SELECT
    (e.first_name || '  ' || e.last_name) employee,
    (m.first_name || '  ' || m.last_name) manager,
    e.job_title
    FROM
   employees e
   LEFT JOIN employees m ON
    m.employee_id = e.manager_id
   ORDER BY
    manager;
```

This query references to the employees table twice: one as e (for employee) and another as m (for manager). The join predicate matches employees and managers using the employee_id and manager_id columns.

The following picture shows the result:

EMPLOYEE	MANAGER	JOB_TITLE
Tommy Bailey		President
Evie Harrison	Ava Sullivan	Sales Representative
Grace Ellis	Ava Sullivan	Sales Representative
Lily Fisher	Ava Sullivan	Sales Representative
Sophia Reynolds	Ava Sullivan	Sales Representative
Sophie Owens	Ava Sullivan	Sales Representative
Poppy Jordan	Ava Sullivan	Sales Representative
Louie Richardson	Blake Cooper	Programmer
Georgia Mills	Callum Jenkins	Shipping Clerk
Maisie Nichols	Callum Jenkins	Shipping Clerk
Eleanor Grant	Callum Jenkins	Shipping Clerk
Hannah Knight	Callum Jenkins	Shipping Clerk
Connor Hayes	Callum Jenkins	Stock Clerk

B) Using Oracle self join to compare rows within the same table example

The following statement finds all employees who have the same hire dates:

```
SQL>SELECT
   e1.hire_date,
  (e1.first_name || ' ' || e1.last_name) employee1,
  (e2.first_name || ' ' || e2.last_name) employee2
FROM
    employees e1
INNER JOIN employees e2 ON
    e1.employee_id > e2.employee_id
    AND e1.hire_date = e2.hire_date
ORDER BY
   e1.hire_date DESC,
   employee1,
employee2;
```

HIRE_DATE	EMPLOYEE1	EMPLOYEE2
07-DEC-16	Rory Kelly	Elliot Brooks
28-SEP-16	Kai Long	Tyler Ramirez
20-AUG-16	Sophia Reynolds	Austin Flores
17-AUG-16	Amelie Hudson	Mohammad Peterson
21-JUN-16	Bella Stone	Ivy Burns
14-JUN-16	Jasmine Hunt	Seth Foster
07-JUN-16	Gracie Gardner	Harper Spencer
07-JUN-16	Rose Stephens	Gracie Gardner
07-JUN-16	Rose Stephens	Harper Spencer
07-JUN-16	Summer Payne	Gracie Gardner
07-JUN-16	Summer Payne	Harper Spencer
07-JUN-16	Summer Payne	Rose Stephens
21-APR-16	Elsie Henry	Matilda Stevens
10-APR-16	Reggie Simmons	Liam Henderson
24-MAR-16	Lucy Crawford	Sienna Simpson
24-MAR-16	Lucy Crawford	Sophie Owens
24-MAR-16	Rosie Morales	Lucy Crawford
24-MAR-16	Rosie Morales	Sienna Simpson
24-MAR-16	Rosie Morales	Sophie Owens
24-MAR-16	Sienna Simpson	Sophie Owens

The e1 and e2 are table aliases for the same employees table.

Oracle GROUP BY

The GROUP BY clause is used in a select statement to group rows into a set of summary rows by values of columns or expressions. The GROUP BY clause returns one row per group.

The GROUP BY clause is often used with aggregate functions such as avg(),count(),max(),min() and sum(). In this case, the aggregate function returns the summary information per group. For example, given groups of products in several categories, the avg() function returns the average price of products in each category, the count() function returns the total number of price of products in each category, the max() function returns the maximum price of products in each category, the min() function returns the minimum price of products in each category, the

sum() function returns the sum of price of products in each category.

The following illustrates the syntax of the Oracle GROUP BY clause:

```
SQL>SELECT
    column_list
   FROM
    T
   GROUP BY c1,c2,c3;
```

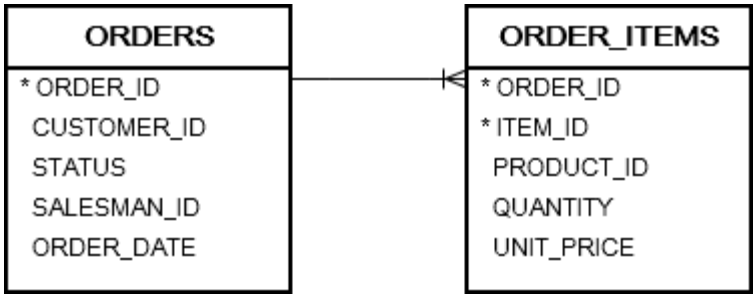
The GROUP BY clause appears after the FROM clause. In case WHERE clause is presented, the GROUP BY clause must be placed after the WHERE clause as shown in the following query:

```
SQL>SELECT
   column_list
   FROM
  T
  WHERE
   condition
 GROUP BY c1, c2, c3;
```

The GROUP BY clause groups rows by values in the grouping columns such as c1, c2 and c3. The GROUP BY clause must contain only aggregates or grouping columns.

Oracle GROUP BY examples

We will use the following orders and order_items in the database for the demonstration:



A) Oracle GROUP BY basic example

The following statement uses the GROUP BY clause to find unique order statuses from the orders table:

```
SQL> SELECT
    status
   FROM
    orders
  GROUP BY
    status;
```

STATUS
Shipped
Pending
Canceled

This statement has the same effect as the following statement that uses the distinct operator:

```
SQL>SELECT
    DISTINCT status
    FROM
    orders;
```

B) Oracle GROUP BY with an aggregate function example

The following statement returns the number of orders by customers:

```
SQL>SELECT
```

```
    customer_id,
    COUNT( order_id )
    FROM
    orders
    GROUP BY
    customer_id
    ORDER BY
    customer_id;
```

CUSTOMER_ID	COUNT(ORDER_ID)
1	4
2	4
3	4
4	4
5	4
6	4
7	4
8	4
9	4

In this example, we grouped the orders by customers and used the function to return the number of orders per group.

To get more meaningful data, we can join the orders table with the customers table as follows:

```
SQL>SELECT
    name,
    COUNT( order_id )
    FROM
    orders
    INNER JOIN customers
        USING(customer_id)
        GROUP BY
         name
        ORDER BY
         name;
```

Here is the result:

NAME	COUNT(ORDER_ID)
AECOM	1
AbbVie	4
Abbott Laboratories	1
Aflac	4
Alcoa	4
American Electric Power	1
AutoNation	4
AutoZone	1
Baker Hughes	1
Bank of New York Mellon Corp.	1

C) Oracle GROUP BY with WHERE clause example

This example uses the GROUP BY clause with a where clause to return the number of shipped orders for every customer:

```
SQL>SELECT
   name,
   COUNT( order_id )
   FROM orders
   INNER JOIN customers USING(customer_id)
   WHERE
   status = 'Shipped'
   GROUP BY
   name
   ORDER BY
   name;
```

Here is the output:

NAME	COUNT(ORDER_ID)
AECOM	1
AbbVie	2
Abbott Laboratories	1
Aflac	2
Alcoa	2
American Electric Power	1
AutoNation	3
AutoZone	1
Baker Hughes	1
Becton Dickinson	1
Bristol-Myers Squibb	1
Centene	2
CenturyLink	4
Community Health Systems	3
DTE Energy	1
Dollar General	1

Note that the Oracle always evaluates the condition in the WHERE clause before the GROUP BY clause.

Oracle HAVING

The HAVING clause is an optional clause of the select statement. It is used to filter groups of rows returned by the group by clause. This is why the HAVING clause is usually used with the GROUP BY clause.

The following illustrates the syntax of the Oracle HAVING clause:

```
SQL>SELECT
    column_list
    FROM
    T
   GROUP BY
    c1
   HAVING
    group_condition;
```

In this statement, the HAVING clause appears immediately after the GROUP BY clause.

If we use the HAVING clause without the GROUP BY clause, the HAVING clause works like the where clause.

Note that the HAVING clause filters groups of rows while the WHERE clause filters rows. This is a main difference between the HAVING and WHERE clauses.

Oracle HAVING clause example

We will use the order_items in the database for the demonstration.

Simple Oracle HAVING example

The following statement uses the GROUP BY clause to retrieve the orders and their values from the order_items table:

```
SQL>SELECT
    order_id,
    SUM( unit_price * quantity ) order_value
   FROM
    order_items
  GROUP BY
    order_id
  ORDER BY
```

```
    order_value DESC;
```

Here is the result:

ORDER_ID	ORDER_VALUE
70	1278962.17
46	1269323.77
78	1198331.59
1	1143716.87
68	1088670.12
27	1084871.49
32	1081679.88
92	1050939.97
59	1043144.72
76	953702.32
104	950118.04
60	926416.51

To find the orders whose values are greater than 1 million, you add a HAVING clause as follows:

```
SQL>SELECT
    order_id,
    SUM( unit_price * quantity ) order_value
   FROM
    order_items
  GROUP BY
    order_id
  HAVING
    SUM( unit_price * quantity ) > 1000000
  ORDER BY
    order_value DESC;
```

The result is:

ORDER_ID	ORDER_VALUE
70	1278962.17
46	1269323.77
78	1198331.59
1	1143716.87
68	1088670.12
27	1084871.49
32	1081679.88
92	1050939.97
59	1043144.72

In this example:

- First, the GROUP BY clause groups orders by their ids and calculates the order values using the sum() function.
- Then, the HAVING clause filters all orders whose values are less than or equal to 1,000,000.

Oracle UNION

The UNION operator is a set operator that combines result sets of two or more select statements into a single result set.

The following illustrates the syntax of the UNION operator that combines the result sets of two queries:

```
SQL>SELECT
    column_list_1
   FROM
    T1
   UNION
   SELECT
    column_list_1
   FROM
    T2;
```

In this statement, the column_list_1 and column_list_2 must have the same number of columns presented in the same order. In

addition, the datatype of the corresponding column must be in the same data type group.

By default, the UNION operator returns the unique rows from both result sets. If we want to retain the duplicate rows, you explicitly use UNION ALL as follows:

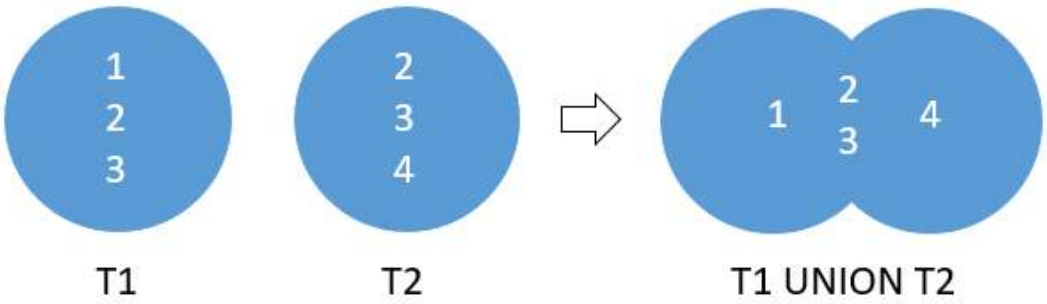
```
SQL>SELECT
    column_list
    FROM
    T1
   UNION ALL
   SELECT
    column_list
   FROM
    T2;
```

Oracle UNION illustration

Suppose, we have two tables T1 and T2:

- T1 has three rows 1, 2 and 3
- T2 also has three rows 2, 3 and 4

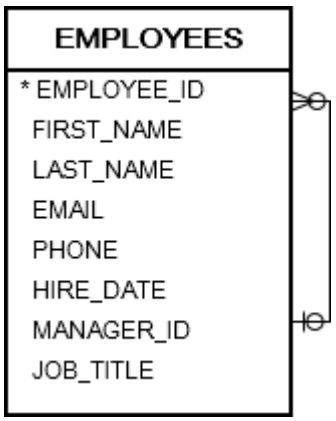
The following picture illustrates the UNION of T1 and T2 tables:



The UNION removed the duplicate rows 2 and 3

Oracle UNION examples

See the following employees and contacts tables in the database.



Oracle UNION example

Suppose, we have to send out emails to the email addresses from both employees and contacts tables. To accomplish this, first, we need to compose a list of email addresses of employees and contacts. And then send out the emails to the list.

The following statement uses the UNION operator to build a list of contacts from the employees and contacts tables:

```
SQL>SELECT
    first_name,
    last_name,
    email,
    'contact'
    FROM
```

```
    contacts
  UNION SELECT
    first_name,
    last_name,
    email,
    'employee'
    FROM
    employees;
```

Here is the result:

FIRST_NAME	LAST_NAME	EMAIL	'CONTACT'
Aaron	Holder	aaron.holder@gilead.com	contact
Aaron	Patterson	aaron.patterson@example.com	employee
Abigail	Palmer	abigail.palmer@example.com	employee
Adah	Myers	adah.myers@dom.com	contact
Adam	Jacobs	adam.jacobs@univar.com	contact
Adrienne	Lang	adrienne.lang@qualcomm.com	contact
Agustina	Conner	agustina.conner@dollartree.com	contact
Al	Schultz	al.schultz@altria.com	contact
Albert	Watson	albert.watson@example.com	employee
Aleshia	Reese	aleshia.reese@adp.com	contact
Alessandra	Estrada	alessandra.estrada@ameriprise.com	contact

Oracle UNION ALL example

The following statement returns the unique last names of employees and contacts:

```
SQL>SELECT
    last_name
    FROM
    employees
    UNION SELECT
    last_name
   FROM
    contacts
```

```
    ORDER BY
    last_name;
```

The query returned 357 unique last names.

LAST_NAME
Abbott
Alexander
Allison
Alston
Arnold
Atkinson
Avila
Bailey
Baldwin
Ball
Barnes
Barnett

However, if we use UNION ALL instead of UNION in the query as follows:

```
SQL>SELECT
    last_name
    FROM
    employees
   UNION ALL SELECT
    last_name
   FROM
    contacts
   ORDER BY
    last_name;
```

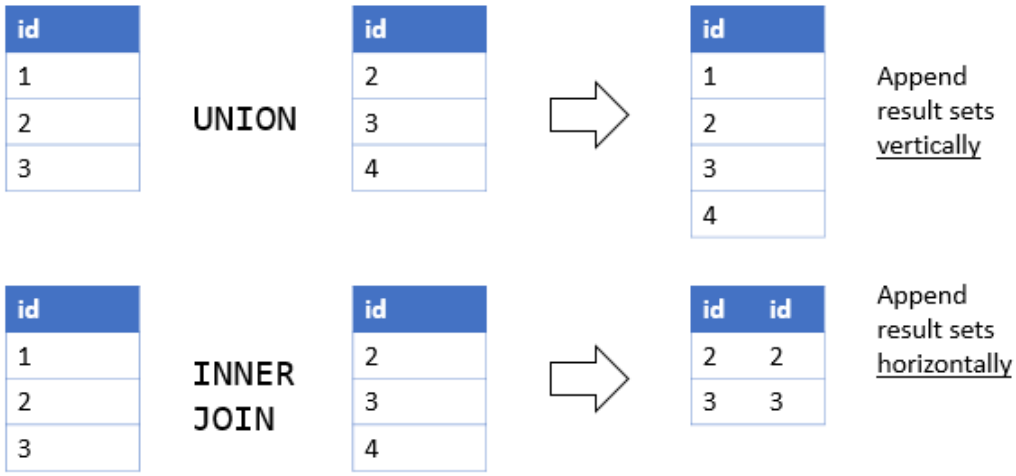
The query returns 426 rows. In addition, some rows are duplicate e.g., Atkinson, Barnett. This is because the UNION ALL operator does not remove duplicate rows.

LAST_NAME
Abbott
Alexander
Allison
Alston
Arnold
Atkinson
Atkinson
Avila
Bailey
Baldwin
Ball
Barnes
Barnett
Barnett
Barrera

Oracle UNION vs. JOIN

A UNION places a result set on top another, meaning that it appends result sets vertically. However, a join such as inner join or left join combines result sets horizontally.

The following picture illustrates the difference between union and join:



Oracle INTERSECT

The Oracle INTERSECT operator compares the result of two queries and returns the distinct rows that are output by both queries.

The following statement shows the syntax of the INTERSECT operator:

```
SQL>SELECT
   column_list_1
  FROM
   T1
  INTERSECT
  SELECT
   column_list_2
  FROM
   T2;
```

Similar to the UNION operator, you must follow these rules when using the INTERSECT operator:

- The number and the order of columns must be the same in the two queries.
- The datatype of the corresponding columns must be in the same data type group.

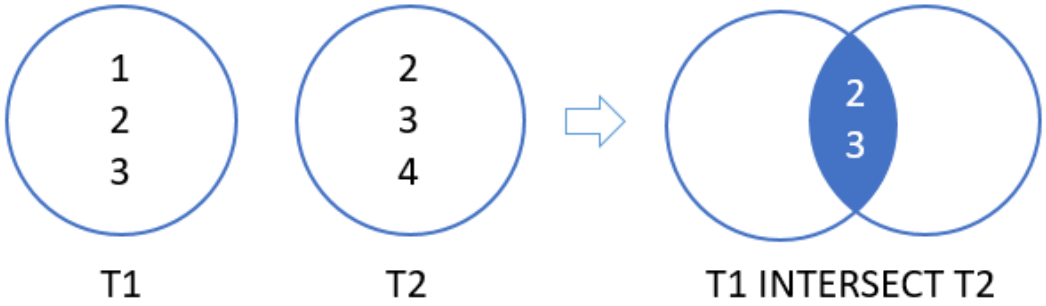
Oracle INTERSECT illustration

Suppose we have two queries that return the T1 and T2 result set.

- T1 result set includes 1, 2, 3.
- T2 result set includes 2, 3, 4.

The intersect of T1 and T2 result returns 2 and 3. Because these are distinct values that are output by both queries.

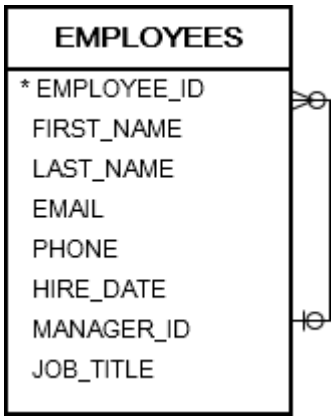
The following picture illustrates the intersection of T1 and T2:



The illustration showed that the INTERSECT returns the intersection of two circles (or sets).

Oracle INTERSECT example

See the following contacts and employees tables in the sample database.



The following statement uses the INTERSECT operator to get the last names used by people in both contacts and employees tables:

```
SQL>SELECT
    last_name
    FROM
    contacts
    INTERSECT
    SELECT
    last_name
    FROM
    employees
    ORDER BY
    last_name;
```

LAST_NAME
Brooks
Bryant
Butler
Cole
Cruz
Ferguson
Flores
Ford
Grant
Hayes
Henderson
Henry
Jordan
Mason

Note that we placed the ORDER BY clause at the last queries to sort the result set returned by the INTERSECT operator.

Oracle MINUS

The Oracle MINUS operator compares two queries and returns distinct rows from the first query that are not output by the

second query. In other words, the MINUS operator subtracts one result set from another.

The following illustrates the syntax of the Oracle MINUS operator:

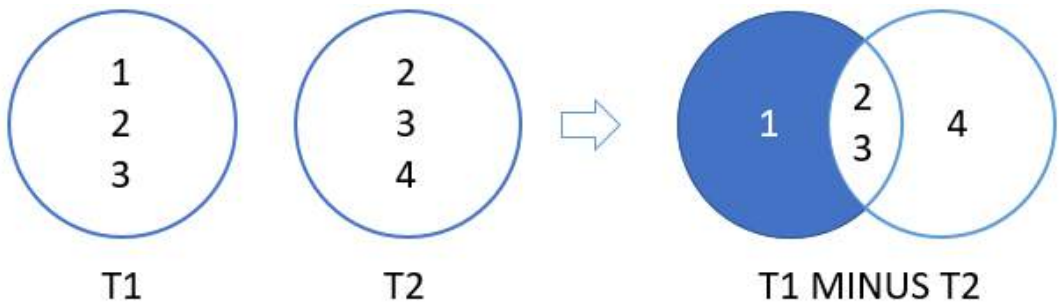
```
SQL>SELECT
    column_list_1
    FROM
    T1
   MINUS
   SELECT
    column_list_2
   FROM
    T2;
```

Similar to the union and intersect operators, the queries above must conform with the following rules:

- The number of columns and their orders must match.
- The datatype of the corresponding columns must be in the same data type group such as numeric or character.

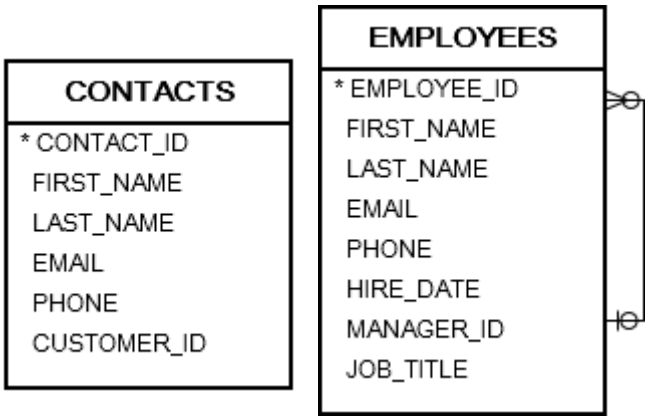
Suppose the first query returns the T1 result set that includes 1, 2 and 3. And the second query returns the T2 result set that includes 2, 3 and 4.

The following picture illustrates the result of the MINUS of T1 and T2:



Oracle MINUS examples

See the following contacts and employees tables in the sample database:



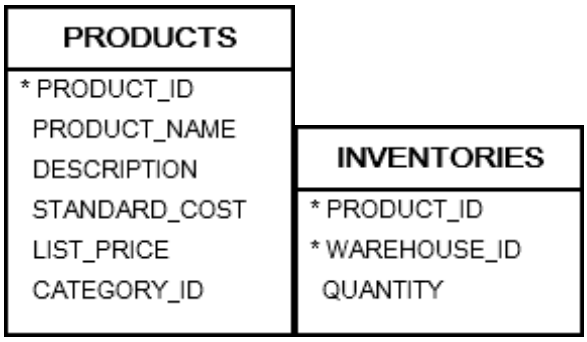
The following statement returns distinct last names from the query to the left of the MINUS operator which are not also found in the right query.

```
SQL>SELECT
    last_name
   FROM
    contacts
   MINUS
    SELECT
    last_name
    FROM
    employees
ORDER BY
    last_name;
```

Here are the last names returned by the first query but are not found in the result set of the second query:

LAST_NAME
Abbott
Allison
Alston
Arnold
Atkinson
Avila
Baldwin
Ball
Barnett
Barrera

See the following products and inventories tables:



The following statement returns a list of product id from the products table, but do not exist in the inventories table:

```
SQL>SELECT
  product_id
 FROM
  products
 MINUS
 SELECT
  product_id
  FROM
  inventories;
```

Here is the result:

PRODUCT_ID
1
10
16
28
45
48
49
51
52
53
55

STUDENTS' LABORATORY ACTIVITY

1. Display the name job, salary for all employees whose job is Clerk or Analyst their salary is not equal to Rs.1000, Rs.3000, Rs.5000.
2. Create a unique listing of all jobs that are in department 30.
3. Write a query to display the name, department number and department name for all employees.
4. Write a query to display the employee name, department name, and location of all employee who earn a commission.
5. Write a query to display the name, job, department number and department name for all employees who works in DALLAS.
6. Write a query to display the number of people with the same job. Save the query @ run it.
7. Create a query to display the employee name and hire date for all employees in same department.
8. Display the employee name and salary of all employees who report to KING.

Integrity Constraints , TCL and DCL

TYPES OF INTEGRITY CONSTRAINTS:

1. Check constraint.
2. Entity Integrity Constraint.
3. Referential integrity constraint

1. **Check Constraint:**The value that each attribute or data item can be assigned is expressed in the form of data type, a range of values or a value from a specified set called as Check constraint. Example: In the relation EMPLOYEE the domain of the attribute Salary may be in the range of 12000 to 300000 or Mark secured by a student in STUDENT relation must be less than or equal to the Total mark.

Creating Check constraint syntax

```
SQL>CREATE TABLE table_name (
   ...
   column_name data_type CHECK (expression),
   ...
);
```

In this syntax, a check constraint consists of the keyword `CHECK` followed by an expression in parentheses. The expression should always involve the column thus constrained. Otherwise, the check constraint does not make any sense.

If we want to assign the check constraint an explicit name, we use the `CONSTRAINT` clause below:

```
CONSTRAINT check_constraint_name
CHECK (expression);
```

Add Check constraint to a table

To add a check constraint to an existing table, we use the alter table add constraint statement as follows:
SQL>ALTER TABLE table_name ADD CONSTRAINT check_constraint_name CHECK(expression);
To drop a check constraint, we use the `ALTER TABLE DROP CONSTRAINT` statement as follows:
SQL> ALTER TABLE table_name DROP CONSTRAINT check_constraint_name;

2. **Entity Integrity Constraint**:The domain values for any attribute that forms a primary key of a relation are validated against the domain constraint, called Entity Integrity Constraint.

It does not to allow null values and redundant values against a primary key.

Adding a primary key to a table in ORACLE:

To add a primary key constraint to an existing table:
SQL>ALTER TABLE table_name ADD CONSTRAINT constraint_name PRIMARY KEY (column1, column2, ...);
Example: SQL>ALTER TABLE vendors ADD CONSTRAINT pk_vendors PRIMARY KEY (vendor_id);
Dropping an Oracle `PRIMARY KEY` constraint

To drop a `PRIMARY KEY` constraint from a table:
SQL>ALTER TABLE table_name DROP CONSTRAINT primary_key_constraint_name;
To drop the primary key constraint of the `vendors` table as follows:
SQL>ALTER TABLE vendors DROP CONSTRAINT pk_vendors;

3.Referential integrity constraint:

- The constraint that the relation R2 must not contain any unmatched foreign key values and it must contain foreign key values matching to the corresponding(Having the same Domain) primary key of another relation R1 to which it refers to, is called as Referentital Integrity Constraint.
- Two constraints while an attempt to update the relations are made:
- (a) We can not delete the records from relation R1 having the matching foreign key values in the relation R2.
- (b) We can not insert records into the relation R2 which is not having a corresponding primary key in the Relation R1.
- For Ex: In the two relations Shipment(SID,PID,QTY) and Supplier(SID, City,Status), the domain of the SID in Supplier and SID in Shipment are same and SID in Shipment is the foreign key referencing to the SID in Supplier .
- Syntax in ORACLE:

```
SQL>CREATE TABLE child_table (
   ...
   CONSTRAINT fk_name
   FOREIGN KEY(col1, col2,...) REFERENCES parent_table(col1,col2) );
```

First, to explicitly assign the foreign key constraint a name, we use the CONSTRAINT clause followed by the name. The CONSTRAINT clause is optional. If we omit it, Oracle will assign a system-generated name to the foreign key constraint.

Second, we specify the FOREIGN KEY clause to define one or more column as a foreign key and parent table with columns to which the foreign key columns reference.

Unlike the primary key constraint, a table may have more than one foreign key constraint.

Transaction Control Language(TCL)

A transaction is a logical unit of work. A transaction begins with an executable SQL statement and ends explicitly with either rollback or commit statements.

Commit : This command is used to successfully end a transaction. It erases all savepoints in the transaction thus releasing all the locks.

Syntax

SQL>commit;

Rollback: This command is used to undo the work done in the current transaction.

Syntax

SQL>rollback;

Data Control Language(DCL)

It provides users with privilege commands. The owner of the database object can allow other database users access to the database .

Grant Privilege command

Syntax

SQL>grant privileges on objectname to username;

Example

SQL>grant select,update on persons to manager;

After successful execution of the above command “grant succeeded” will be displayed and it grants privileges like select and update on persons to manager.

Revoke Privilege command

Syntax

SQL>revoke privileges on objectname from username;

Example

SQL>revoke select,update on persons from manager;

After successful execution of the above command “revoke succeeded” will be displayed and it revokes all the privileges like select and update on persons from manager.

STUDENTS' LABORATORY ACTIVITY

1. Create a student database table using create command using Regd. No as Primary Key , insert data of your class.

2. Create a supplier database table using sid as Primary Key. Insert 5 records.

3. Create a part database table using pid as Primary Key . Insert 5 records.

4. Create a shipment database table using combination of sid and pid as Primary Key, sid as a foreign key referencing to the sid in supplier table and pid as a foreign key referencing to the pid in part table . Insert 5 records.

5. Practice of all Data Retrieval, DML, DDL, TCL and DCL commands used in Oracle by writing queries.