Unit-3

Open Source Database: MySQL- Introduction - setting up account-starting, terminating and writing your own SQL programs, record selection technology, working with strings - date and time, sorting query results.

MySQL- Introduction

MySQL Database

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons −

- MySQL is released under an open-source license. So you have nothing to pay to use it.
- MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and works well even with large data sets.
- MySQL is very friendly to PHP, the most appreciated language for web development.
- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
- MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

Installing MySQL on Linux/UNIX

The recommended way to install MySQL on a Linux system is via RPM. MySQL AB makes the following RPMs available for download on its website −

- **MySQL** − The MySQL database server manages the databases and tables, controls user access and processes the SQL queries.
- **MySQL-client** − MySQL client programs, which make it possible to connect to and interact with the server.
- **MySQL-devel** − Libraries and header files that come in handy when compiling other programs that use MySQL.
- **MySQL-shared** − Shared libraries for the MySQL client.

- **MySQL-bench** − Benchmark and performance testing tools for the MySQL database server.

The MySQL RPMs listed here are all built on a **SuSE Linux system**, but they will usually work on other Linux variants with no difficulty.

Now, you will need to adhere to the steps given below, to proceed with the installation −

- Login to the system using the **root** user.
- Switch to the directory containing the RPMs.
- Install the MySQL database server by executing the following command. Remember to replace the filename in italics with the file name of your RPM.

The above command takes care of installing the MySQL server, creating a user of MySQL, creating necessary configuration and starting the MySQL server automatically.

You can find all the MySQL related binaries in /usr/bin and /usr/sbin. All the tables and databases will be created in the /var/lib/mysql directory.

The following code box has an optional but recommended step to install the remaining RPMs in the same manner −

```
[root@host]# rpm -i MySQL-client-5.0.9-0.i386.rpm
[root@host]# rpm -i MySQL-devel-5.0.9-0.i386.rpm
[root@host]# rpm -i MySQL-shared-5.0.9-0.i386.rpm
[root@host]# rpm -i MySQL-bench-5.0.9-0.i386.rpm
```

Installing MySQL on Windows

The default installation on any version of Windows is now much easier than it used to be, as MySQL now comes neatly packaged with an installer. Simply download the installer package, unzip it anywhere and run the setup.exe file.

The default installer setup.exe will walk you through the trivial process and by default will install everything under C:\mysql.

Test the server by firing it up from the command prompt the first time. Go to the location of the **mysqld server** which is probably C:\mysql\bin, and type −

```
mysqld.exe --console
```

If all went well, you will see some messages about startup and **InnoDB**. If not, you may have a permissions issue. Make sure that the directory that holds your data is accessible to whatever user (probably MySQL) the database processes run under.

MySQL will not add itself to the start menu, and there is no particularly nice GUI way to stop the server either. Therefore, if you tend to start the server by double clicking the mysqld executable, you should remember to halt the process by hand by using mysqladmin, Task List, Task Manager, or other Windows-specific means.

Verifying MySQL Installation

After MySQL, has been successfully installed, the base tables have been initialized and the server has been started: you can verify that everything is working as it should be via some simple tests.

Use **mysqladmin** binary to check the server version. This binary would be available in /usr/bin on linux and in C:\mysql\bin on windows.

```
[root@host]# mysqladmin --version
```

It will produce the following result on Linux. It may vary depending on your installation −

```
mysqladmin  Ver 8.23 Distrib 5.0.9-0, for redhat-linux-gnu on i386
```

If you do not get such a message, then there may be some problem in your installation and you would need some help to fix it.

You can connect to your MySQL server through the MySQL client and by using the **mysql** command. At this moment, you do not need to give any password as by default it will be set as blank.

You can just use following command −

```
[root@host]# mysql
```

It should be rewarded with a mysql> prompt. Now, you are connected to the MySQL server and you can execute all the SQL commands at the mysql> prompt as follows −

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
|  mysql   |
|  test    |
+----------+
2 rows in set (0.13 sec)
```

Post-installation Steps

MySQL ships with a blank password for the root MySQL user. As soon as you have successfully installed the database and the client, you need to set a root password as given in the following code block −

```
[root@host]# mysqladmin -u root password "new_password";
```

Now to make a connection to your MySQL server, you would have to use the following command −

```
[root@host]# mysql -u root -p
Enter password:*******
```

UNIX users will also want to put your MySQL directory in your PATH, so you won't have to keep typing out the full path everytime you want to use the command-line client.

For bash, it would be something like −

```
export PATH = $PATH:/usr/bin:/usr/sbin
```

Running MySQL at Boot Time

If you want to run the MySQL server at boot time, then make sure you have the following entry in the /etc/rc.local file.

```
/etc/init.d/mysqld start
```

Also,you should have the mysqld binary in the /etc/init.d/ directory.

Running and Shutting down MySQL Server

First check if your MySQL server is running or not. You can use the following command to check it −

```
ps -ef | grep mysqld
```

If your MySql is running, then you will see **mysqld** process listed out in your result. If server is not running, then you can start it by using the following command −

```
root@host# cd /usr/bin
./safe_mysqld &
```

Now, if you want to shut down an already running MySQL server, then you can do it by using the following command −

```
root@host# cd /usr/bin
./mysqladmin -u root -p shutdown
Enter password: ******
```

Setting Up a MySQL User Account

For adding a new user to MySQL, you just need to add a new entry to the **user** table in the database **mysql**.

The following program is an example of adding a new user **guest** with SELECT, INSERT and UPDATE privileges with the password **guest123;** the SQL query is −

```
root@host# mysql -u root -p
Enter password:*******
mysql> use mysql;
Database changed

mysql> INSERT INTO user
   (host, user, password,
   select_priv, insert_priv, update_priv)
   VALUES ('localhost', 'guest',
   PASSWORD('guest123'), 'Y', 'Y', 'Y');
Query OK, 1 row affected (0.20 sec)
```

```
mysql> FLUSH PRIVILEGES;
Query OK, 1 row affected (0.01 sec)

mysql> SELECT host, user, password FROM user WHERE user = 'guest';
+-----------+---------+------------------+
|   host    |  user   |     password     |
+-----------+---------+------------------+
| localhost |  guest  | 6f8c114b58f2ce9e |
+-----------+---------+------------------+
1 row in set (0.00 sec)
```

When adding a new user, remember to encrypt the new password using PASSWORD() function provided by MySQL. As you can see in the above example, the password mypass is encrypted to 6f8c114b58f2ce9e.

Notice the FLUSH PRIVILEGES statement. This tells the server to reload the grant tables. If you don't use it, then you won't be able to connect to MySQL using the new user account at least until the server is rebooted.

You can also specify other privileges to a new user by setting the values of following columns in user table to 'Y' when executing the INSERT query or you can update them later using UPDATE query.

Another way of adding user account is by using GRANT SQL command. The following example will add user **zara** with password **zara123** for a particular database, which is named as **TUTORIALS**.

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use mysql;
Database changed

mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
   -> ON TUTORIALS.*
   -> TO 'zara'@'localhost'
   -> IDENTIFIED BY 'zara123';
```

This will also create an entry in the MySQL database table called as **user**.

NOTE − MySQL does not terminate a command until you give a semi colon (;) at the end of the SQL command.

The /etc/my.cnf File Configuration

In most of the cases, you should not touch this file. By default, it will have the following entries −

```
[mysqld]
datadir = /var/lib/mysql
socket = /var/lib/mysql/mysql.sock
```

```
[mysql.server]
user = mysql
basedir = /var/lib

[safe_mysqld]
err-log = /var/log/mysqld.log
pid-file = /var/run/mysqld/mysqld.pid
```

Here, you can specify a different directory for the error log, otherwise you should not change any entry in this table.

Administrative MySQL Command

Here is the list of the important MySQL commands, which you will use time to time to work with MySQL database −

USE Databasename − This will be used to select a database in the MySQL workarea.

SHOW DATABASES − Lists out the databases that are accessible by the MySQL DBMS.

SHOW TABLES − Shows the tables in the database once a database has been selected with the use command.

SHOW COLUMNS FROM ***tablename:*** Shows the attributes, types of attributes, key information, whether NULL is permitted, defaults, and other information for a table.

My SQL PHP Syntax

MySQL works very well in combination of various programming languages like PERL, C, C++, JAVA and PHP. Out of these languages, PHP is the most popular one because of its web application development capabilities.

.

PHP provides various functions to access the MySQL database and to manipulate the data records inside the MySQL database. You would require to call the PHP functions in the same way you call any other PHP function.

The PHP functions for use with MySQL have the following general format −

mysqli *function*(value,value,...);

The second part of the function name is specific to the function, usually a word that describes what the function does. The following are two of the functions, which we will use in our tutorial −$mysqli = new mysqli($dbhost, $dbuser, $dbpass, $dbname);

mysqli->query(,"SQL statement");

The following example shows a generic syntax of PHP to call any MySQL function.

```
<html>
<head>
    <title>PHP with MySQL</title>
  </head>

  <body>
    <?php
       $retval = mysqli - > function(value, [value,...]);
       if( !$retval ) {
          die ( "Error: a related error message" );
       }
       // Otherwise MySQL  or PHP Statemen?>
 </body>
</html>
```

Starting from the next chapter, we will see all the important MySQL functionality along with PHP.

SHOW INDEX FROM tablename − Presents the details of all indexes on the table, including the PRIMARY KEY.

SHOW TABLE STATUS LIKE tablename\G − Reports details of the MySQL DBMS performance and statistics.

MySQL SELECT Statement

The SELECT statement in MySQL is used to **fetch data from one or more tables**. We can retrieve records of all fields or specified fields that match specified criteria using this statement. It can also work with various scripting languages such as PHP, Ruby, and many more.

SELECT Statement Syntax

It is the most commonly used SQL query. The general syntax of this statement to fetch data from tables are as follows:

1. **SELECT** field_name1, field_name 2,... field_nameN
2. **FROM** table_name1, table_name2...
3. [**WHERE** condition]
4. [**GROUP BY** field_name(s)]
5. [**HAVING** condition]
6. [**ORDER BY** field_name(s)]
7. [OFFSET M][LIMIT N];

Syntax for all fields:

1. **SELECT * FROM** tables [**WHERE** conditions]
2. [**GROUP BY** fieldName(s)]
3. [**HAVING** condition]
4. [**ORDER BY** fieldName(s)]
5. [OFFSET M][LIMIT N];

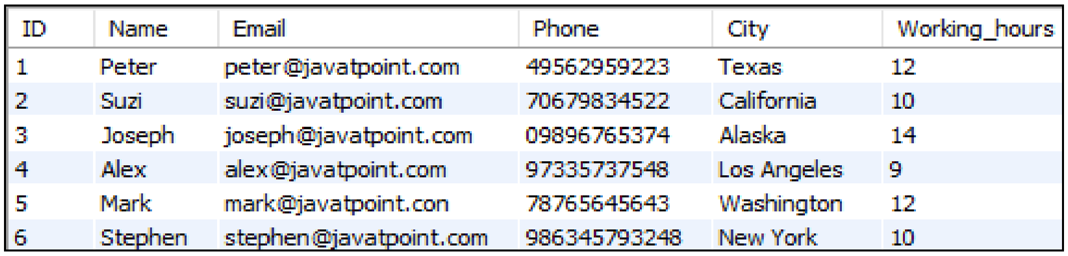
Parameter Explanation

The SELECT statement uses the following parameters:

Parameter Name	**Descriptions**
field_name(s) or *	It is used to specify one or more columns to returns in the result set. The asterisk (*) returns all fields of a table.
table_name(s)	It is the name of tables from which we want to fetch data.
WHERE	It is an optional clause. It specifies the condition that returned the matched records in the result set.
GROUP BY	It is optional. It collects data from multiple records and grouped them by one or more columns.
HAVING	It is optional. It works with the GROUP BY clause and returns only those rows whose condition is TRUE.
ORDER BY	It is optional. It is used for sorting the records in the result set.
OFFSET	It is optional. It specifies to which row returns first. By default, It starts with zero.
LIMIT	It is optional. It is used to limit the number of returned records in the result set.

MySQL SELECT Statement Example:

Let us understand how SELECT command works in MySQL with the help of various examples. Suppose we have a table named **employee_detail** that contains the following data:

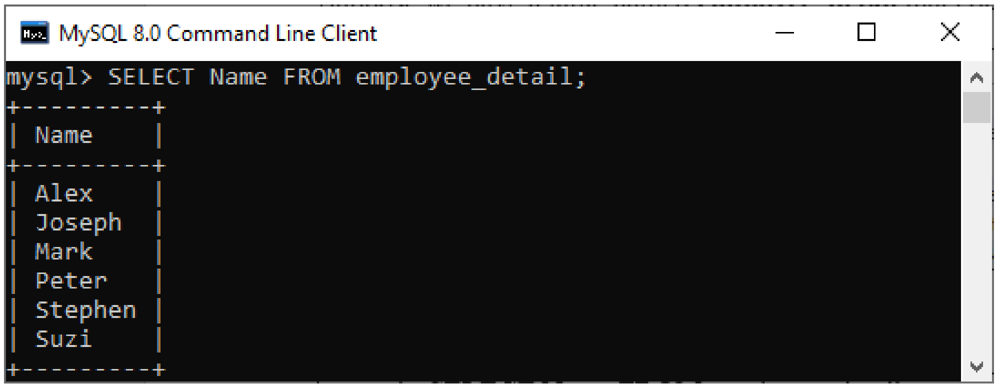


ID	Name	Email	Phone	City	Working_hours
1	Peter	peter@javatpoint.com	49562959223	Texas	12
2	Suzi	suzi@javatpoint.com	70679834522	California	10
3	Joseph	joseph@javatpoint.com	09896765374	Alaska	14
4	Alex	alex@javatpoint.com	97335737548	Los Angeles	9
5	Mark	mark@javatpoint.con	78765645643	Washington	12
6	Stephen	stephen@javatpoint.com	986345793248	New York	10

1. If we want to retrieve a **single column from the table**, we need to execute the below query:

1. mysql> **SELECT Name FROM** employee_detail;

We will get the below output where we can see only one column records.

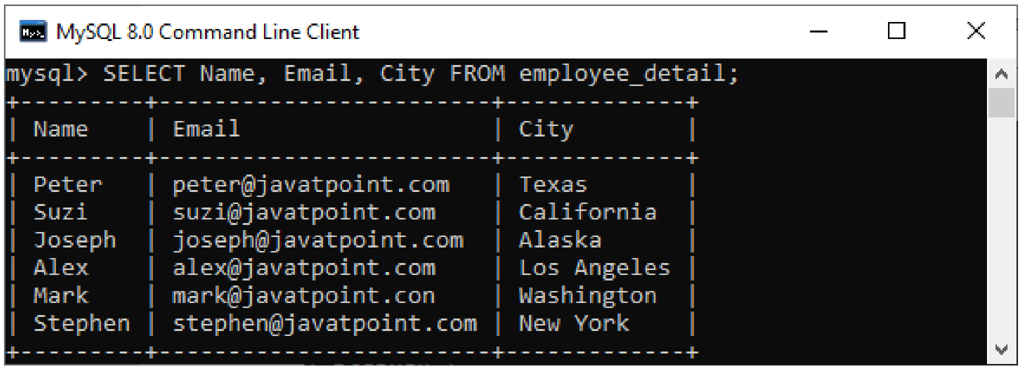


```
mysql> SELECT Name FROM employee_detail;
+---------+
| Name    |
+---------+
| Alex    |
| Joseph  |
| Mark    |
| Peter   |
| Stephen |
| Suzi    |
+---------+
```

2. If we want to query **multiple columns from the table**, we need to execute the below query:

1. mysql> **SELECT Name**, Email, City **FROM** employee_detail;

We will get the below output where we can see the name, email, and city of employees.

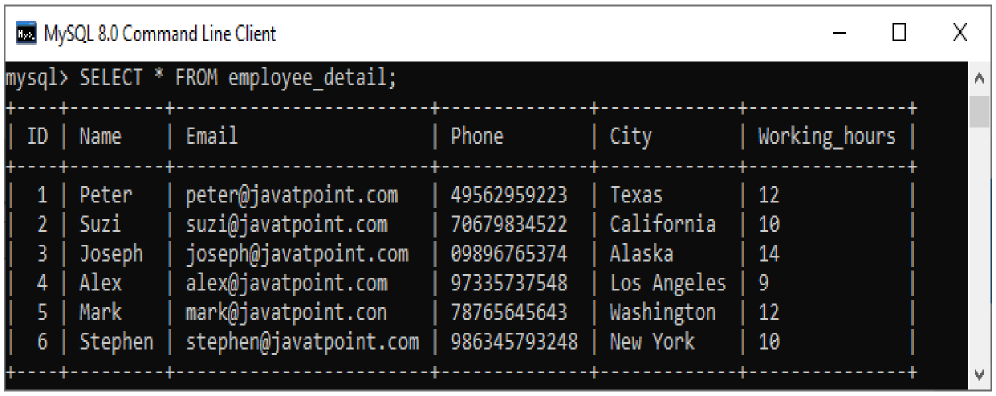


```
mysql> SELECT Name, Email, City FROM employee_detail;
+---------+------------------------+-------------+
| Name    | Email                  | City        |
+---------+------------------------+-------------+
| Peter   | peter@javatpoint.com   | Texas       |
| Suzi    | suzi@javatpoint.com    | California  |
| Joseph  | joseph@javatpoint.com  | Alaska      |
| Alex    | alex@javatpoint.com    | Los Angeles |
| Mark    | mark@javatpoint.con    | Washington  |
| Stephen | stephen@javatpoint.com | New York    |
+---------+------------------------+-------------+
```

3. If we want to fetch data from **all columns of the table**, we need to use all column's names with the select statement. Specifying all column names is not convenient to the user, so MySQL uses an **asterisk** (*) to retrieve all column data as follows:

1. mysql> **SELECT** * **FROM** employee_detail;

   We will get the below output where we can see all columns of the table.



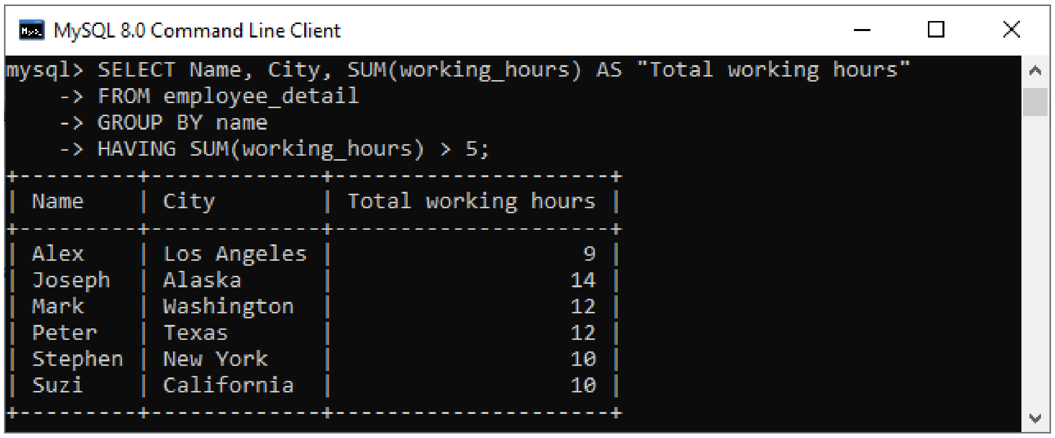
MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM employee_detail;
+----+---------+----------------------+--------------+-------------+---------------+
| ID | Name    | Email                | Phone        | City        | Working_hours |
+----+---------+----------------------+--------------+-------------+---------------+
|  1 | Peter   | peter@javatpoint.com | 49562959223  | Texas       | 12            |
|  2 | Suzi    | suzi@javatpoint.com  | 70679834522  | California  | 10            |
|  3 | Joseph  | joseph@javatpoint.com| 09896765374  | Alaska      | 14            |
|  4 | Alex    | alex@javatpoint.com  | 97335737548  | Los Angeles | 9             |
|  5 | Mark    | mark@javatpoint.con  | 78765645643  | Washington  | 12            |
|  6 | Stephen | stephen@javatpoint.com | 986345793248 | New York  | 10            |
+----+---------+----------------------+--------------+-------------+---------------+
```

4. Here, we use the **SUM function** with the **HAVING** clause in the SELECT command to get the employee name, city, and total working hours. Also, it uses the **GROUP BY** clause to group them by the Name column.

1. **SELECT Name**, City, SUM(working_hours) **AS** "Total working hours"
2. **FROM** employee_detail
3. **GROUP BY Name**
4. **HAVING** SUM(working_hours) > 5;

   It will give the below output:

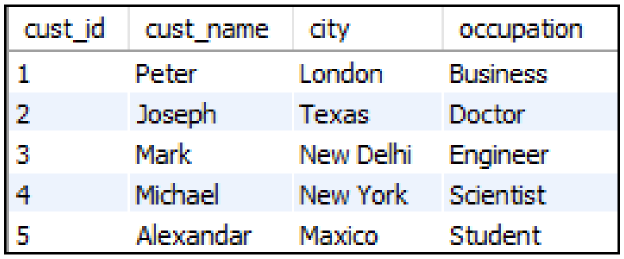


MySQL 8.0 Command Line Client

```
mysql> SELECT Name, City, SUM(working_hours) AS "Total working hours"
    -> FROM employee_detail
    -> GROUP BY name
    -> HAVING SUM(working_hours) > 5;
+---------+-------------+---------------------+
| Name    | City        | Total working hours |
+---------+-------------+---------------------+
| Alex    | Los Angeles |                   9 |
| Joseph  | Alaska      |                  14 |
| Mark    | Washington  |                  12 |
| Peter   | Texas       |                  12 |
| Stephen | New York    |                  10 |
| Suzi    | California  |                  10 |
+---------+-------------+---------------------+
```

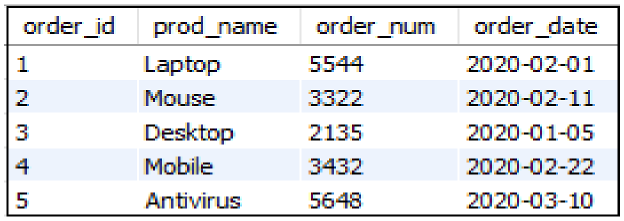
5. MySQL SELECT statement can also be used to retrieve records from multiple tables by using a **JOIN statement**. Suppose we have a table named **"customer"** and **"orders"** that contains the following data:

Table: customer



cust_id	cust_name	city	occupation
1	Peter	London	Business
2	Joseph	Texas	Doctor
3	Mark	New Delhi	Engineer
4	Michael	New York	Scientist
5	Alexandar	Maxico	Student

Table: orders



order_id	prod_name	order_num	order_date
1	Laptop	5544	2020-02-01
2	Mouse	3322	2020-02-11
3	Desktop	2135	2020-01-05
4	Mobile	3432	2020-02-22
5	Antivirus	5648	2020-03-10

Execute the following SQL statement that returns the matching records from both tables using the **INNER JOIN** query:

```
SELECT cust_name, city, order_num, order_date
FROM customer INNER JOIN orders
ON customer.cust_id = orders.order_id
WHERE order_date < '2020-04-30'
ORDER BY cust_name;
```

Sorting Query Results

Use the ORDER BY clause to order the rows selected by a query. Sorting by position is useful in the following cases:

- To order by a lengthy select list expression, you can specify its position in the ORDER BY clause rather than duplicate the entire expression.
- For compound queries containing set operators UNION, INTERSECT, MINUS, or UNION ALL, the ORDER BY clause must specify positions or aliases rather than explicit expressions. Also, the ORDER BY clause can appear only in the last component query. The ORDER BY clause orders all rows returned by the entire compound query.

The mechanism by which Oracle Database sorts values for the ORDER BY clause is specified either explicitly by the NLS_SORT initialization parameter or implicitly by the NLS_LANGUAGE initialization parameter. You can change the sort mechanism dynamically from one linguistic sort sequence to another using the ALTER SESSION statement. You can also specify a specific sort sequence for a single query by using the NLSSORT function with the NLS_SORT parameter in the ORDER BY clause.

Joins

A **join** is a query that combines rows from two or more tables, views, or materialized views. Oracle Database performs a join whenever multiple tables appear in the FROM clause of the query. The select list of the query can select any columns from any of these tables. If any two of these tables have a column name in common, then you must qualify all references to these columns throughout the query with table names to avoid ambiguity.

Join Conditions

Most join queries contain at least one **join condition**, either in the FROM clause or in the WHERE clause. The join condition compares two columns, each from a different table. To execute a join, Oracle Database combines pairs of rows, each containing one row from each table, for which the join condition evaluates to TRUE. The columns in the join conditions need not also appear in the select list.

To execute a join of three or more tables, Oracle first joins two of the tables based on the join conditions comparing their columns and then joins the result to another table based on join conditions containing columns of the joined tables and the new table. Oracle continues this process until all tables are joined into the result. The optimizer determines the order in which Oracle joins tables based on the join conditions, indexes on the tables, and, any available statistics for the tables.

IA WHERE clause that contains a join condition can also contain other conditions that refer to columns of only one table. These conditions can further restrict the rows returned by the join query.

Equijoins

An **equijoin** is a join with a join condition containing an equality operator. An equijoin combines rows that have equivalent values for the specified columns. Depending on the internal algorithm the optimizer chooses to execute the join, the total size of the columns in the equijoin condition in a single table may be limited to the size of a data block minus some overhead. The size of a data block is specified by the initialization parameter DB_BLOCK_SIZE.

Self Joins

A **self join** is a join of a table to itself. This table appears twice in the FROM clause and is followed by table aliases that qualify column names in the join condition. To perform a self join, Oracle Database combines and returns rows of the table that satisfy the join condition.

Cartesian Products

If two tables in a join query have no join condition, then Oracle Database returns their **Cartesian product**. Oracle combines each row of one table with each row of the other. A Cartesian product always generates many rows and is rarely useful. For example, the Cartesian product of two tables, each with 100 rows, has 10,000 rows. Always include a join condition unless you specifically need a Cartesian product. If a query joins three or more tables and you do not specify a join condition for a specific pair, then the optimizer may choose a join order that avoids producing an intermediate Cartesian product.

Inner Joins

An **inner join** (sometimes called a **simple join**) is a join of two or more tables that returns only those rows that satisfy the join condition.

Outer Joins

An **outer join** extends the result of a simple join. An outer join returns all rows that satisfy the join condition and also returns some or all of those rows from one table for which no rows from the other satisfy the join condition.

- To write a query that performs an outer join of tables A and B and returns all rows from A (a **left outer join**), use the LEFT [OUTER] JOIN syntax in the FROM clause, or apply the outer join operator (+) to all columns of B in the join condition in the WHERE clause. For all rows in A that have no matching rows in B, Oracle Database returns null for any select list expressions containing columns of B.
- To write a query that performs an outer join of tables A and B and returns all rows from B (a **right outer join**), use the RIGHT [OUTER] JOIN syntax in the FROM clause, or apply the outer join operator (+) to all columns of A in the join condition in the WHERE clause. For all rows in B that have no matching rows in A, Oracle returns null for any select list expressions containing columns of A.
- To write a query that performs an outer join and returns all rows from A and B, extended with nulls if they do not satisfy the join condition (a **full outer join**), use the FULL [OUTER] JOIN syntax in the FROM clause.

You can use outer joins to fill gaps in sparse data. Such a join is called a **partitioned outer join** and is formed using the *query_partition_clause* of the *join_clause* syntax. Sparse data is data that does not have rows for all possible values of a dimension such as time or department. For example, tables of sales data typically do not have rows for products that had no sales on a given date. Filling data gaps is useful in situations where data sparsity complicates analytic computation or where some data might be missed if the sparse data is queried directly.

Oracle recommends that you use the FROM clause OUTER JOIN syntax rather than the Oracle join operator. Outer join queries that use the Oracle join operator (+) are subject to the following rules and restrictions, which do not apply to the FROM clause OUTER JOIN syntax:

- You cannot specify the (+) operator in a query block that also contains FROM clause join syntax.
- The (+) operator can appear only in the WHERE clause or, in the context of left-correlation (that is, when specifying the TABLE clause) in the FROM clause, and can be applied only to a column of a table or view.
- If A and B are joined by multiple join conditions, then you must use the (+) operator in all of these conditions. If you do not, then Oracle Database will return only the rows resulting from a simple join, but without a warning or error to advise you that you do not have the results of an outer join.
- The (+) operator does not produce an outer join if you specify one table in the outer query and the other table in an inner query.
- You cannot use the (+) operator to outer-join a table to itself, although self joins are valid. For example, the following statement is **not** valid:

```
-- The following statement is not valid:
SELECT employee_id, manager_id
  FROM employees
  WHERE employees.manager_id(+) = employees.employee_id;

```

  However, the following self join is valid:

```
SELECT e1.employee_id, e1.manager_id, e2.employee_id
  FROM employees e1, employees e2
  WHERE e1.manager_id(+) = e2.employee_id;
```

- The (+) operator can be applied only to a column, not to an arbitrary expression. However, an arbitrary expression can contain one or more columns marked with the (+) operator.
- A WHERE condition containing the (+) operator cannot be combined with another condition using the OR logical operator.
- A WHERE condition cannot use the IN comparison condition to compare a column marked with the (+) operator with an expression.
- A WHERE condition cannot compare any column marked with the (+) operator with a subquery.

If the WHERE clause contains a condition that compares a column from table B with a constant, then the (+) operator must be applied to the column so that Oracle returns the rows from table A for which it has generated nulls for this column. Otherwise Oracle returns only the results of a simple join.

In a query that performs outer joins of more than two pairs of tables, a single table can be the null-generated table for only one other table. For this reason, you cannot apply the (+) operator to columns of B in the join condition for A and B and the join condition for B and C. Please refer to SELECT for the syntax for an outer join.

Antijoins

An antijoin returns rows from the left side of the predicate for which there are no corresponding rows on the right side of the predicate. That is, it returns rows that fail to match (NOT IN) the subquery on the right side.

Semijoins

A semijoin returns rows that match an EXISTS subquery without duplicating rows from the left side of the predicate when multiple rows on the right side satisfy the criteria of the subquery.

Semijoin and antijoin transformation cannot be done if the subquery is on an OR branch of the WHERE clause.

Using Subqueries

A **subquery** answers multiple-part questions. For example, to determine who works in Taylor's department, you can first use a subquery to determine the department in which Taylor works. You can then answer the original question with the parent SELECT statement. A subquery in the FROM clause of a SELECT statement is also called an **inline view**. A subquery in the WHERE clause of a SELECT statement is also called a **nested subquery**.

A subquery can contain another subquery. Oracle Database imposes no limit on the number of subquery levels in the FROM clause of the top-level query. You can nest up to 255 levels of subqueries in the WHERE clause.

If columns in a subquery have the same name as columns in the containing statement, then you must prefix any reference to the column of the table from the containing statement with the table name or alias. To make your statements easier to read, always qualify the columns in a subquery with the name or alias of the table, view, or materialized view.

Oracle performs a **correlated subquery** when a nested subquery references a column from a table referred to a parent statement any number of levels above the subquery. The parent statement can be a SELECT, UPDATE, or DELETE statement in which the subquery is nested. A correlated subquery is evaluated once for each row processed by the parent statement. Oracle resolves unqualified columns in the subquery by looking in the tables named in the subquery and then in the tables named in the parent statement.

A correlated subquery answers a multiple-part question whose answer depends on the value in each row processed by the parent statement. For example, you can use a correlated subquery to determine which employees earn more than the average salaries for their departments. In this case, the correlated subquery specifically computes the average salary for each department.

Use subqueries for the following purposes:

- To define the set of rows to be inserted into the target table of an INSERT or CREATE TABLE statement
- To define the set of rows to be included in a view or materialized view in a CREATE VIEW or CREATE MATERIALIZED VIEW statement
- To define one or more values to be assigned to existing rows in an UPDATE statement
- To provide values for conditions in a WHERE clause, HAVING clause, or START WITH clause of SELECT, UPDATE, and DELETE statements
- To define a table to be operated on by a containing query

  You do this by placing the subquery in the FROM clause of the containing query as you would a table name. You may use subqueries in place of tables in this way as well in INSERT, UPDATE, and DELETE statements.

  Subqueries so used can employ correlation variables, but only those defined within the subquery itself, not outer references. Please refer to *table_collection_expression* for more information.

  Scalar subqueries, which return a single column value from a single row, are a valid form of expression. You can use scalar subquery expressions in most of the places where *expr* is called for in syntax. Please refer to "Scalar Subquery Expressions" for more information.