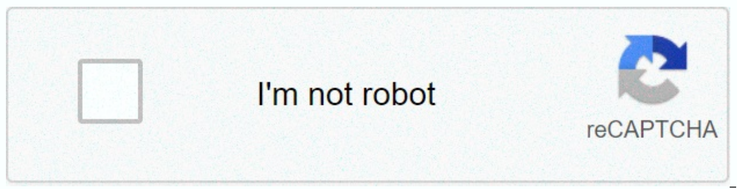

Continue

Sql tutorial advanced with examples pdf

SQL is incredibly powerful, and like any well-made development tool, it has some commands that it's important for a good developer to know. Here is a list of SQL queries that are really important for coding & optimization. Each of the queries in our SQL tutorial is consistent to almost any system that interacts with a SQL database. 1.SQL Query for Downloading Tables This query can be run to retrieve the list of tables that are in a database where the database is My_Schema. The select command allows users to define the columns they want to retrieve in query output. This command is also useful for retrieving which column users want to see as the output table. The SELECT statement is applied to pick data from a table. Data retrieved is added to a result table, named the result set. The output is saved in a result table. This output table is also referred to as the result set. SELECT * FROM My_Schema.Tables; 2. Query to select columns from a table This is perhaps the most used SQL queries example. In the example below, we are extracting Student_ID column or attribute from the STUDENT table. The Select statement is used to select data from the database. CHOOSE STUDENT_ID FROM STUDENT; To view all attributes from a particular table, this is the right question to use: SELECT * FROM STUDENT; 3. Query for data output Using a constraint This SQL query retrieves the specified attributes from the table on the Employee ID =0000 SELECT EMP_ID, NAME FROM EMPLOYEE_TBL WHERE EMP_ID = '0000'; 4. Query for Output Sorted Data Use 'Order By' This query orders the results with respect to the attribute referenced to use Order By - so for example if that attribute is an integer data type, then the result would either be sorted in ascending or descending order; likewise if the data type is one String then the result would be ordered in alphabetical order. The order by statement is used to sort data from the table. The order after statement should always be used in the last of the SQL query. SELECT EMP_ID, LAST_NAME FROM EMPLOYEE WHERE THE LOCALITY = 'Seattle' ORDER BY EMP_ID; The order of the result can also be set manually, using asc for ascending and desc for descending. Ascending (ASC) is the default condition for the ORDER BY statement. In other words, if users do not enter ASC or DESC after the column name, then the result will be ordered in ascending order only. SELECT EMP_ID, LAST_NAME FROM EMPLOYEE_TBL WHERE CITY = 'INDIANAPOLIS' ORDER BY EMP_ID ASC; 5.SQL Query for Output Sorted Data With 'Group by' The 'Group By' property groups the resulting data according to the specified attribute. The SQL query below will select Name, Age columns from Patients table, then will filter them by Age value to include records where Age is more than 40 then will group items with similar age value and then finally will eject them sorted by Name. The basic rule is that the group after statement should always follow a where statement in a Select statement and must precede the Order by statement. CHOOSE Name, Age from Patients VAR Age > 40 GROUP BY Name, Age ORDER BY Name; Another selection of use of Group After: this expression will select items with a price less than 70 from the Order table, will group items with a similar price, will sort the output by price and will also add the column COUNT(price) which will show how many items with similar price were found: SELECT COUNT(price), price FROM order WHERE PRICE < 70 GROUP BY PRICE ORDER THROUGH PRICE NOTE! you should use much the same set of columns for both SELECT and GROUP BY commands, otherwise you will get an error. Many thanks to Sachidannad for pointing out! SQL queries for Data manipulation Using mathematical functions there are a lot of built-in mathematical functions like COUNT and AVG that provide basic functions for counting the number of results and averaging them respectively. 6. Data manipulation Using COUNTIDS This query shows the total number of customers by counting each customer ID. In addition, it groups the results according to each customer's country. In count, if users define DISTINCT, then they cal also define query_partition_clause. This clause is part of the analytical clause, and the other clauses order_by_clause and windowing_clause are not allowed. Syntax: SELECT COUNT(carbon name) FROM table name, SELECT COUNT(CustomerID), Country FROM Customers GROUP BY COUNTRY; 7. Data manipulation Using SUM, the sum of the attribute given to it is calculated as an argument. SUM is an aggregate function and it calculates the sum of all distinct values. and the sum of all duplicate values. SELECT SUM(Salary)FROM Employee WHERE Emp_Age < 30; 8. Data manipulation Using AVG Simple – an average of a given attribute. Average is also an aggregate function in SQL. The AVG() function calculates the average of non-NULL values in a column. It ignores the null values. SELECT AVG(Price)FROM Products; 9.SQL Query for Listing All Views SQL Query lists all views available in the schema. * My_Schema FROM SELECT FROM views. 10. Query to create a view a view is a custom table formed as a result of a query. It has tables and rows just like any other table. It is usually a good idea to run queries in SQL as independent views because this allows them to be downloaded later to view the query results, instead of calculating the same command each time for a specific set of results. CREATE VIEW FAILING_STUDENTS AS SELECT S_NAME, Student_ID FROM STUDENT WHERE GPA > 40; 11. Query to get a Show Default syntax for selecting attributes from a table is views as well. SELECT * FROM FROM 12. Query to update a view This query updates the view named 'Product List' – and if this view does not exist, then the Product List view may be created as specified in this query. The view is also called as a virtual table. In other words, displaying is just a mirrored copy of a table whose data is the result of a stored query. A view is a legitimate copy of another table or sequence of tables. A view obtains its information or data from the tables from previously created tables called base tables. Base tables are real tables. Any procedure implemented on a view really changes the base table. Users can use views just like the actual or base tables. In the view, users can use various DDL, DML commands to update, insert, and delete. CREATE OR REPLACE VIEW [Product List] AS SELECT ProductID, ProductName, Category FROM PRODUCTS WHERE Expired = No; 13. Query for Drop a View This query will drop or delete a view named 'V1'. The important thing to remember here is that DROP VIEW is disallowed if there are any views depending on the view you are about to drop. IVVY V1; 14. Query to display user tables A user-defined table is a representation of defined information in a table, and can be used as an argument for procedures or user-defined functions. Because they are so useful, it is useful to keep track of them using the following question. User tables explain the current user's relationship tables. SELECT * FROM Sys.object WHERE Type='u' 15. Question to Primary Key Display A primary key uniquely identifies all values within a table. A primary key does not entail a NULL constraint and a unique constraint in a declaration. In other words, it prevents different rows from having similar values or sequences of columns. It does not allow null values. The primar key can be defined on a single column or the combination of two columns in a table. It is responsible for all relationships between the tables. The following SQL query lists all fields in a table's primary key. SELECT * from Sys.Objects WHERE Type='PK' 16. Question for Unique Key Display A unique key allows a column to ensure that all of its values are different. Unique key also recognizes another tupppel uniquely in relation to or table. A table can have more than one unique key. Unique key constraints can only take a NULL value for the column. SELECT * FROM Sys.Object WHERE Type='uq' 17. Show Foreign Keys Foreign Keys link one table to another — they are attributes in one table that refer to the primary key in another table. SELECT * FROM Sys.Object WHERE Type='f' Primary, Unique, and Foreign are part of the limitations of SQL. Limitations are critical to data scalability, compliance, and sincerity. Restrictions implement specific rules, ensure the data comply with the described. These are, for example, the imposed on the columns of the database table. These are applied to limit the type of data in the table. This assures the efficiency and authenticity of the database. 18. Trigger display A trigger is sort of an 'event listener' — i.e. a trigger. The list of defined triggers can be displayed using the following query. SELECT * FROM Sys.Object WHERE Type='tr' 19. View internal tables Internal tables are formed as one of the product of a user action and are usually not accessible. The data in internal tables cannot be manipulated; however, the internal table metadata can be displayed using the following query. SELECT * FROM Sys.Objects WHERE Type='it' 20. Displaying a list of procedures A stored procedure is a group of advanced SQL queries that logically form a single unit and perform a specific task. Thus, using the following question you can keep track of them: CHOOSE * FROM Sys.Objects WHERE Type ='p' Improve your skills with SQL trainer (for beginners and advanced developers) Try SQL Trainer for free.. and TWENTY More Advanced SQL Queries for Our Users! 21. Changing the values of two columns in a table In this and subsequent examples, we will use a common enterprise database including multiple tables that are easily visualized. Our practice DB will include a Customer Table and an Order table. The Customers table will contain some obvious columns including ID, Name, Address, zip, and email, for example, where we assume for now that the primary key field for indexing is Customer_ID field. With this in mind, we can easily imagine an Orders table that also contains the indexed customer ID field, along with details of each order that the customer makes. This table will contain the Order Number, Quantity, Date, Item, and Price. In our first of sql examples, imagine a situation where the mail and phone fields were transposing and all phone numbers were incorrectly entered in the zip code field. We can easily fix this problem with the following SQL statement: UPDATE Customers SET Zip=Phone, Phone=Zip 22. Return a column of unique values now, suppose that our data entry operator added the same Customers to the Customers table more than once by mistake. As you know, correct indexing requires that the key field contains only unique values. To fix this issue, we will use SELECT DISTINCT to create an indexable list of unique customers: SELECT DISTINCT ID FROM Customers 23. To make a Top 25 with SELECT TOP Clause Next, imagine that our Customers table has grown to include thousands of entries, but we just want to show a selection of 25 of these entries to demonstrate the column headings and the SELECT TOP clause allows us to enter the number of entries to return, as a Top-25 list. In this example, we will return the top 25 from our Customers table: TOP 25 FROM CUSTOMERS WHERE Customer_ID<>NULL; 24. Searching for SQL Tables with Wildcard's Wildcard characters or operators as %makes it easy to find specific strings in a large table with thousands of entries. Suppose we want to find all our customers who have names starting at Herb including Herberts, and Herbertson. The symbol % wildcard can be used to achieve such a result. The following SQL query returns all rows from the Customer table where the field Customer_name begins with Herb: SELECT * From customers WHERE Name AS 'Herb%' 25. Between Monday and Tuesday Today is Wednesday, and we come to work and find that our new data entry clerk in training has entered all new orders incorrectly on Monday and Tuesday. We want to teach our new intern to find and correct all incorrect records. What is the easiest way to get all records from the Orders table in on Monday and Tuesday? The middle kit turns the task into a breeze: SELECT The ID FROM Order WHERE DATE BETWEEN '01/12/2018' AND '01/13/2018' 26. Finding the intersection of two tables Without a doubt the whole reason why a relational database exists in the first place is to find matching records in two tables! In the JOIN statement, you accomplish this core goal with SQL and make the task simple. Here we will get a list of all records that have matches in the Customers and Orders tables: SELECT ID FROM Customers INNER JOIN ORDERS ON Customers.ID = Orders.ID The point of INNER JOIN, in this case, is to select records in the Customers table that have a matching customer ID values in the Orders table and only return those records. Of course, there are many types of JOIN, such as FULL, SELF, and LEFT, but for now, let's keep things interesting and move on to more different types of SQL queries. 27. Doubling power with UNION We can combine the results of two SQL queries example to one naturally with the union keyword. Let's say we want to create a new table by combining Customer_name and phone from Customers with a list of that customer's latest orders so we can look for patterns and perhaps suggest future purchases. Here's a quick way to accomplish the task: CHOOSE PHONE FROM Customers UNION SELECT ITEMS FROM ORDER The UNION keywords allow you to combine JOINS and other criteria to achieve very powerful new table generation potential. 28. Making Column Labels More Friendly Aliasing Column Labels gives us the convenience of renaming a column label to something more readable. There is a trade-off when naming columns to make them concise results in reduced readability in subsequent daily use. In our Orders table, the item column contains the description of purchased products. Let's see how to alias the article column to temporarily rename it for greater ease of use: SELECT Item that item_description FROM Order 29. and Everywhere! Wouldn't it be great if there was a set of conditions you could depend on each time? SQL queries using ANY and ALL can make this ideal a reality! Let's look at how the all-keyword is used to only include records when a set of conditions is true for ALL records. In the following example, we will return records from the Orders table where the idea is to get a list of high volume orders for a given item, in this case for customers who ordered more than 50 of the product: SELECT Item FROM Order WHERE id = ALL (SELECT ID FROM Order WHERE Quantity > 50) 30. Writing Developer Friendly SQL An often overlooked but very important feature of SQL scripts is to add comments to a script of queries to explain what it does for the benefit of future developers who may need to revise and update your SQL queries. A SQL script is a collection of SQL elements and commands accumulated as a file in SQL Scripts. This script file can include many SQL commands or PL/SQL codes. You can use SQL Scripts to build, edit, design, execute, and delete files. The — single row and the /* .. */ multi-line separators allow us to add useful comments to scripts, but this is also used in another valuable way. Sometimes a section with code may not be used, but we don't want to delete it, because we anticipate using it again. Here we can simply add the comment twig to disable the moment: /* This question below commented on so it will not execute */ /* SELECT ITEM FROM ORDER WHERE DATE ALL = (SELECT Order_ID FROM Order WHERE QUANTITY > 50) */ /* The SQL query below will be executed ignoring the text by -- */SELECT object -- single comment FROM Order -- another single comment WHERE id ALL = (SELECT ID FROM Order WHERE Quantity > 25) 31. SQL queries for Database Management So far, we've explored SQL query samples to query tables and combine records from multiple queries. Now it's time to step up and look at the database at a structural level. Let's start with the simplest SQL statement of anyone who creates a new database. Here we will create DB as a container for our Customers and Order tables used in the previous ten examples above: CREATE DATABASE AllSales 32. Adding tables to Our new DB Next, we will actually add the Customers table that we have used in previous examples, and then add some of the column labels that we are already familiar with: CREATE TABLE Customers (ID varchar(80), Name

varchar(80), Phone varchar(20), ...); Although most databases are created using a user interface such as Access or OpenOffice, it is important to know how to create and delete databases and tables programmatically via code with SQL statements. This is especially so when you install a new web app and the UI asks new users to enter names DBs to be added during installation. 33. Change and remove the SQL ALTER statement is used to change or change the meaning of a table. For the relationship tables with columns, the ALTER statement is used to update the table to the new or changed rules or definition. Alter belongs to the DDL category of commands. Data definition languages can be described as a pattern for commands by which data structures are represented. Imagine that you decide to send a birthday card to your customers to show your appreciation for their business, and so you want to add a birthday field to the Customers table. In these SQL examples, you can see how easy it is to modify existing tables using the ALTER TABLE Customers ADD Birthday varchar(80) If a table is corrupted with bad data, you can quickly delete it like this: DROP TABLE table_name 34. The key to successful indexing An index is a schema element that contains a record for each content that arrives in the indexed column of the database table or cluster and provides a high-speed path to rows. There are many types of indexes such as Bitmap Indexes, Partitioned Indexes, Feature-Based Indexes, and Domain Indexes. Exact indexing requires that the Primary Key column contains only unique values for this purpose. This ensures that JOIN statements will maintain integrity and produce valid matches. Let's create our Customers table again and establish the ID column as the primary key: CREATE TABLE Customers (ID int NOT NULL, Name varchar(80) NOT NULL, PRIMARY KEY (ID)); We can extend the functionality of the primary key to automatically step up from a base. Change the ID record above to add AUTO_INCREMENT as in the following statement: ID int NOT NULL AUTO_INCREMENT 35. Advanced concepts to improve performance When practical, it is always better to type the column name list in a SELECT statement rather than using the * separator as a wildcard to select all columns. SQL Server has to do a search and replace operation to find all the columns in your table and enter them in the statement for you (each time SELECT is executed). For example: SELECT* FROM Customers Would actually perform much faster on our database as: SELECT Name, Birthday, Phone, Address, Zip FROM Customers Performance pitfalls can be avoided in many ways. For example, avoid the time sinkhole to force SQL Server to check the system/master database each time by using only one stored procedure name, and never prefix it with SP_. Setting NOCOUNT ON also reduces the time required for SQL Server to count rows that are affected by INSERT, DELETE, and other commands. Using INNER JOIN with a condition is much faster than using WHERE statements with conditions. We advise developers to learn SQL server queries to an advanced level for this purpose. For production purposes, these tips can be crucial for adequate Note that our guidance examples tend to tend to gynna den INRE JOIN. 36. Villkorliga delquery resultat SQL-operatorn FINNS tester för förekomsten av poster i en underquery och returnerar ett värde SANT om en underquery returnerar en eller flera poster. Ta en titt på den här frågan med ett underfrågavillkor: Välj Namn från kunder där det finns (Välj Artikel från order där Customers.ID = Orders.ID OCH Pris < 50)= in= this= example= above,= the= select= returns= a= value= of= true= when= a= customer= has= orders= valued= at= less= than= $50.= 37.= copying= selections= from= table= to= table= there= are= a= hundred= and= one= uses= for= this= sql= tool.= suppose= you= want= to= archive= your= yearly= orders= table= into= a= larger= archive= table.= this= next= example= shows= how= to= do= it.= insert= into= yearly_orders= select= *= from= orders= where=> <=1 018= this= example= will= add= any= records= from= the= year= 2018= to= the= archive.= 38. catching= null= results= the= null= is= the= terminology= applied= to= describe= an= absent= value.= null= does= not= mean= zero.= a= null= value= in= a= column= of= a= table= is= a= condition= in= a= domain= that= seems= to= be= empty.= a= column= with= a= null= value= is= a= domain= with= absent= value.= it= is= essential= to= recognize= that= a= null= value= is= distinct= from= a= zero.= in= cases= where= null= values= are= allowed= in= a= field,= calculations= on= those= values= will= produce= null= results= as= well.= this= can= be= avoided= by= the= use= of= the= ifnull= operator.= in= this= next= example,= a= value= of= zero= is= returned= rather= than= a= value= of= null= when= the= calculation= encounters= a= field= with= a= null= value := select= item,= price= *= (qtyinstock= += ifnull(qtyonorder,= 0))= from= orders= 39. having= can= be= relieving!= the= problem= was= that= the= sql= where= clause= could= not= operate= on= aggregate= functions.= the= problem= was= solved= by= using= the= having= clause.= as= an= example,= this= next= query= fetches= a= list= of= customers= by= the= region= where= there= is= at= least= one= customer= per= region:= select= count(id),= region= from= customers= group= by= region= having= count(id)=>0; 40. Bind upp saker med Strängar! Låt oss ta en titt på bearbetning av innehållet i fältdata med hjälp av funktioner. Substräng är förmodligen den mest värdefulla av alla inbyggda funktioner. Det ger dig en del av kraften i Regex, men det är inte så komplicerat som Regex. Anta att du vill hitta delsträngen till vänster om punkterna i en webbadress. Så här gör du det med en SQL Select-fråga: SELECT SUBSTRING_INDEX(www.bytescout.com, ., 2); Denna linje kommer att returnera allt till vänster om den andra förekomsten av . och så, i detta fall kommer det att återvända www.bytescout.com Kontrollera den här videon för att lära sig om varje SQL-fråga: .. och 20 mer SQL Queries example !! 41. Use COALESCE to return the first non-null expression SQL Coalesce used to manage the null values of the database. In this method, the NULL values are replaced with the user-defined value. The SQL Coalesce function assesses the parameters in series and always delivers the first non-null</=1> non-null</=1> from the specified argument record. Syntax SELECT COALESCE(NULL,NULL,'ByteScout',NULL,'Byte') Output ByteScout 42. Use Convert to convert any value to a specific data type This is used to convert a value to a defined data type. For example, if you want to convert a specific value to int data type then the convert function can be used to achieve this. For example, Syntax SELECT CONVERT(int, 27.64) Output 27 43. DENSE_RANK()Analytical query It is an analytical query that calculates the rank of a row in an ordered collection of rows. An output rank is a number starting from 1. DENSE_RANK is one of the most important analytical SQL queries. It returns rank preferences as sequential numbers. It does not jump rank in case of relationships. For example, the following question will give sequential joint sensiment to the employee. SELECT eno, dno, salary, DENSE_RANK() OVER (PARTITION BY dno ORDER BY SALARY) AS EMPLOYEE RANKING; ENO DNO SALARY RANKINGS ---------- ---------- ---------- ---------- 7933 10 1500 1 7788 10 2650 2 7831 10 6000 3 7362 20 900 1 7870 20 1200 2 7564 20 2575 3 7784 20 4000 4 7903 20 4000 4 7901 30 550 1 7655 30 1450 2 7522 30 1450 2 7844 30 1700 3 7493 30 1500 4 7698 30 2850 5 44. Query_partition_clause The query_partition_clause breaks the output amount into deployments, or collections, of data. The development of the analytical query is limited to the limits forced by these partitions, related to the process a GROUP BY statement changes the performance of an aggregate function. If query_partition_clause is eliminated, the entire output collection is interpreted as a separate partition. The following query applies an OVER statement, so the average displayed is based on all records of the output set. CHOOSE eno, dno, salary, AVG(salary) OVER () AS avg_sal FROM employee; EO DNO SALARY AVG_SAL ---------- ---------- ---------- ---------- 7364 20 900 2173,21428 7494 30 1700 2173,21428 7522 30 1350 2173,21428 7567 20 20 20 2 03075 2173.21428 7652 30 1350 2173.21428 7699 30 2950 2173.21428 7783 10 2550 2173214 2 8 7789 20 3100 2173,21428 7838 10 5100 2173,21428 7845 30 1600 2173,21428 7877 20 1200 2 173.21428 7901 30 1050 2173.21428 7903 20 3100 2173.21428 7935 10 1400 2173.21428 45. Finding the last five records from the Now table, if you want to retrieve the last eight records from the table, it's always difficult to get that data if your table contains huge information. For example, you want to get the last 8 entries fr on the employees table then you can use rownum and a union clause. The line is temporarily in SQL. For example, select * from Employee A where rownum <=8 union selects * from (Select * from Employee An order after rowid desc) where rownum <=8; The above SQL query will give you the last eight entries from the employees' table where rownum is a pseudocolumn. It indexes data in an output set. 46. The LAG LAG is applied in order to obtain data from a Line. This is an analytical function. For example, the following query gives the pay from the previous line to calculate the difference between the pay for the current line and the one for the previous line. This query applies the ORDER BY OF THE LAG function. The default is 1 unless you define offset. The arbitrary default condition is given if the offset is moved past the window's area. The default is null unless you define the default. Syntax SELECT dtno, eno, emname, job, pay, LAG(sal, 1, 0) OVER (PARTITION BY dtno ORDER BY SALARY) AS salary_prev FROM EMPLOYEE; Output DTNO ENO ENAME JOBB SAL SAL_PREV ---------- ---------- ---------- --------- ---------- ---------- 10 7931 STEVE CLERK 1300 0 10 7783 JOHN MANAGER 2450 1300 10 7834 KING PRESIDENT 2450 1300 10 7834 KING PRESIDENT 15000 2450 20 7364 ROBIN ROBIN CLERK 800 0 20 7876 BRIAN CLERK 11 000 800 20 7567 SHANE MANAGER 2975 1 100 20 7784 SCOTT ANALYST 3000 2975 20 7908 KANE ANALYST 3000 3000 3000 30 7900 JAMES CLERK 950 0 30 7651 CONNER SALESMAN 1250 950 30 7522 MATTHEW SALESMAN 1250 1250 30 7843 VIVIAN SALESMAN 1500 1250 30 7494 ALLEN SALESMAN 1600 1500 30 7695 GLEN MANAGER 2850 1600 47. LEAD LEAD is also an analytical query that is applied to retrieve data from rows extra down the output set. The following query gives the pay from the next line to calculate the discrepancy between the pay of the current line and the subsequent line. The default is 1 unless you define offset. The arbitrary default condition is given if the offset is moved past the window's area. The default is null unless you define the default. SELECT eno, empname, job, salary, LEAD(pay, 1, 0) OVER (ORDER BY salary) AS salary_next, LEAD(salary, 1, 0) OVER (ORDER BY salary) - salary AS salary_diff FROM employee; ENO EMPNAME JOBS SALARY SALARY_NEXT SALARY_DIFF ---------- ---------- --------- ---------- ---------- ---------- 7369 STEVE CLERK 800 950 150 7900 JEFF CLERK 950 1100 150 7876 ADAMS CLERK 1100 1250 150 7521 JOHN SALESMAN 1250 1250 0 7654 MARK SALESMAN 1250 1300 50 7934 TANTO CLERK 1300 1500 200 000 7844 MATT SALESMAN 1500 1600 100 7499 ALEX Salesman 1600 2450 850 7782 BOON MANAGER 2450 2850 400 7698 BLAKE MANAGER 2850 2975 125 7566 JONES MANAGER 2975 3000 25 7788 SCOTT ANALYST 3000 3000 0 7902 FORD ANALYST 3000 5000 2000 7839 KING PRESIDENT 5000 0 -5000 48. PERCENT_RANK The PERCENT_RANK analytical question. The ORDER BY statement is necessary for this issue. Excluding a partitioning clause from the OVER statement, the entire output set determines a separate partition. The first line of the standardized set is entered 0, and the last row of the set is specified 1. Examples of the SQL query provide the following output. Syntax SELECT prdid, SUM(amount), PERCENT_RANK() OVER (ORDER BY SUM(AMOUNT) DESC) AS percent_rank FROM SALES GROUP BY PRDID ORDER BY PRDID; Outgoing PRDID SUM(AMOUNT) PERCENT_RANK ---------- ---------- 1 22623,5 0 2 2 1 49. MY Use an empty OVER clause converts MIN into an analytical function. This is also an analytical question. This interprets the entire result set as a single partition. It gives you the lowest salary for all employees and their original tasks. For example, the following query shows the use of MIN in Query Select. SELECT eno, empname, dtno, pay, MIN(pay) OVER (PARTITION BY dtno) AS min_result FROM employee; ENO EMPNAME DTNO SALARY MIN_RESULT ---------- ---------- ---------- ---------- ---------- 7782 CLARK 10 2450 1300 7839 KING 10 5000 1300 7934 MILLER 10 1300 1300 7566 JONES 20 2975 800 7902 FORD 20 3000 800 7876 ADAMS 20 1100 800 7369 SMITH 20 800 800 7788 SCOTT 20 3000 800 7521 TITLE 30 1250 950 7844 TURNER 30 1500 950 7499 ALLEN 30 1600 950 7900 JAMES 30 950 950 7698 BLAKE 30 2850 950 7654 MARTIN 30 1250 950 50. MAX Using an empty line over statement converts MAX into an analytical function. The absence of a partitioning statement indicates the entire output set is interpreted as a separate partition. This gives the maximum salary for all employees and their original tasks. For example, the following query shows the use of MAX in the select query. SELECT eno, empname, dtno, salary, MAX(pay) OVER () AS max_result FROM employee; ENO EMPNAME DTNO SALARY MAX_RESULT ---------- ---------- ---------- ---------- ---------- 7369 SMITH 20 800 3000 7499 ALLEN 30 1600 3000 7521 TITLE 30 1250 3000 75 66 JONES 20 2975 3000 7654 MARTIN 30 1250 3000 7698 BLAKE 30 2850 3000 7782 CLARK 10 2450 30 00 7788 SCOTT 20 3000 3000 7839 KING 10 5000 3000 7844 TURNER 30 1500 3000 7876 ADAMS 20 20 0 1100 3000 7900 JAMES 30 950 3000 7902 FORD 20 3000 3000 7934 MILLER 10 1300 3000 51. Top-N queries Top-N queries provide a process to limit the number of rows delivered from organized assemblages of data. They are remarkably beneficial when users want to give the top or bottom number of rows from a table. For example, the following query gives the 20 lines with 10 different values: SELECT PRICE FROM sales_order ORDER BY Price; PRICE ---------- 100 100 200 200 300 300 400 400 500 500 600 PRICE ---------- 600 700 700 800 800 900 900 1000 1000 20 lines selected. 52. CORR Analytic Query The CORR analytic function is utilized to determine the correlation coefficient. This query is also used to calculate the Pearson correlation coefficient. The function calculates the following on rows in the table without null values. This query always returns the values between +1 and -1, which describes the following: Syntax: CORR(exp1, exp2) [OVER (analytic_clause)] Example SELECT empid, name, dno, pay, job, CORR(SYSDATE - joiningdate, pay) OVER () AS my_corr_val FROM employee; 53. NTILE Analytic Query NTILE enables users to divide a sequence set into a detailed number of relatively similar groups, or containers, deletion sanctioning. If the number rows in the collection are smaller than the number of containers defined, the number of containers will be reduced. The basic syntax is as shown below: NTILE(exp) OVER ([partition_clause] order_by) Example SELECT empid, name, dno, pay, NTILE(6) OVER (ORDER BY pay) AS container_no FROM employee; 54. VARIANCE, VAR_POP and VAR_SAMP Query The VARIANCE, VAR_POP and VAR_SAMP are aggregate functions. These are used to determine the variance, group deviation and variance of a collection of data individually. As aggregate queries or functions, they reduce the number of rows, therefore the expression aggregate. If the data is not organized, we change the total rows in the Employee table to a separate row of the aggregate values. Example: The following query shows the use of these features: SELECT VARIANCE(payroll) AS var_salary, VAR_POP(payroll) AS pop_salary, VAR_SAMP(payroll) AS samp_salary FROM employee; VAR_SALARY POP_SALARY SAMP_SALARY ------------ ----------- ------------ 1479414.97 1588574.81 1388717.27 55. STDDEV, STDDEV_POP and STDDEV_SAMP Questions StDDEV, STDDEV_POP and STDDEV_SAMP aggregate questions or functions are applied to determine the standard deviation, population standard deviation, and cumulative sample standard deviation individually. As aggregate queries, they reduce the number of rows, therefore the expression aggregate. If the data is not ordered, we convert all rows in the EMPLOYEE table to a separate row. For example, the following query shows SELECT STDDEV(payroll) AS stddev_salary, STDDEV_POP(pay) AS pop_salary, STDDEV_SAMP(pay) AS samp_salary FROM employee; STDDEV_SALARY POP_SALARY SAMP_SALARY ---------- ---------- ---------- 1193.50 1159588 1193603 If there is more than one account after the null has been dropped, the STDDEV function gives the result of STDDEV_SAMP. Using an empty OVER statement, the STDDEV query converts the result into an analytical query. The absence of a partitioning indicates the entire output set is interpreted as a particular partition, so we accept the standard deviation of the pay and the primary tasks. 56. Pattern Matching Pattern Matching Pattern Matching Syntax adds different options. Data must be processed accurately and in a correct form. The BY and ORDER BY criteria partition for all SQL analytical queries are applied to divide the data into accumulations and within each group. If no partitions are specified, it is considered the entire sequence set is a huge partition. For example, the measure statement specifies the column result that will be provided for each match. Syntax measures STRT.tstamp AS initial_tstamp, LAST(UP.tstamp) AS first_tstamp, LAST(DOWN.tstamp) AS finished_tstamp Example DEFINE UP AS UP. PRODUCTS_SOLD > PREV(UP.products_sold), FLAT AS FLAT.products_sold = DOWN AS DOWN. PRODUCTS_SOLD < PREV(DOWN.products_sold) 57. FIRST_VALUE The easiest way to get analytical functions is to start by studying aggregate functions. An aggregate function collects or collects data from many rows into a unique result line. For example, users can apply the AVG function to get an average on all salaries in the EMPLOYEE table. Let's take a look at how First_Value can be used. The primary explanation for FIRST_VALUE analytical function is shown below. Syntax: FIRST_VALUE { (pronuncication) [NULLS] | (pronunciator [NULLS]) } OVER (analytical clause) Example SELECT eno, dno, salary, FIRST_VALUE(pay) IGNORE NULLS OVER (PARTITION BY dno ORDER BY SALARY) AS lowest_salary_in_dept FROM employee; The above query will ignore the null values. 58. LAST_VALUE The primary explanation for LAST_VALUE analytical question or function is shown below. Syntax: LAST_VALUE { (pronuncication) [{ NULLS] | (pr / NULLS]) OVER (analytical statement) LAST_VALUE analytical question is related to the LATEST analytical function. This feature enables users to retrieve the last output from an organized column. Applying default window output can be surprising. For example, SELECT eno, dno, pay, LAST_VALUE(pay) IGNORE NULLS OVER (PARTITION BY dno ORDER BY PAY) AS highest_salary_in_dept FROM EMPLOYEE; 59. Prediction The design test foretelles the gender and age of clients who are expected to most adopt a contract card (case = 1). The PREDICTION function takes the price matrix correlated with the design and applies for marital status, and house size as predictors. The PREDICTION function syntax can also use a piece of arbitrary grouping information when you receive a partitioned model. SELECT CLIENT_GENDER, COUNT(*) AS CT, ROUND(AVG(age)) AS AVERAGE_AGE FROM MINING_DATA_SHOP WHERE PREDICTION(EXAMPLE COST MODEL USE client_marital_status, house_size) = 1 GROUP BY CLIENT_GENDER ORDER BY CLIENT_GENDER; CUST_GENDER CNT AVG_AGE ---------- ---------- ---------- F 270 40 M 585 41 60. CLUSTER_SET CLUSTER_SET can get data in one of the pair steps: It can use a mining type object to the data, or it can break the data by performing an analytical clause that creates and uses one or more moving mining patterns. This example lists the properties that have the greatest influence on cluster deployment for client ID 1000. The query requests the CLUSTER_DETAILS and CLUSTER_SET, which use the my_sample clustering model. Example SELECT S.cluster_id, probe, CLUSTER_DETAILS(my_sample, S.cluster_id, 7 USE T.*) kset FROM (SELECT v.*, CLUSTER_SET(my_sample, USE *) nset from mining_data WHERE client_id = 1000) T, TABLE(T.nset) Q ORDER THROUGH 2 DESC; A cluster is group tables that distribute the corresponding data blocks, i.e. all the tables are actually together. For example, EMPLOYEE and DEPARTMENT tables are connected to the DNO column. If you them, it will actually store all rows in the same data block. About the author ByteScout Team of Writers ByteScout has a team of professional writers skilled in various technical topics. We select the best authors to cover interesting and trendy topics for our readers. We love developers and we hope our articles will help you learn about programming and programmers. TRY PDF.CO ONLINE APPS AND WEB API TweetShareWhatsAppPinLinkedIn TweetShareWhatsAppPinLinkedIn