SQL

03 May 2024 16:57

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

Quick ref: add/or, alter, update, between-and, limit-offset ,joins, select -into, insert into-select,top,like-pattern,constraints,delete,drop,truncate,create view-as, create index-on, exists ,group by-having, order by,in,union,union all,where,ifnull n coalesce,instr(colnme,char)---->first occurance,substr,concat,replace(colname,char1,char2)

Tid-by default index, not visible in help tablename(which gives details abt table)

```
Create table tablename;
Insert into tablename values(…..);
Select column  from tablename;
Select column, length(column) from tablename;
Drop table tablename;
Alter table table name rename old_columnname to new_columnname;
```

DATABASE MANAGEMENT SYSTEM

- ✧ Database--------> organized collection of data(types :- relational, distributed, cloud,object-oriented, data warehouses(repository of data),NO sql,graph,OLTP,
- ✧ Any database which satisfies 12 rules by codd's rule it is called rdbms
- ✧ ACID Properties
  - Atomicity - all or nothing
  - Consistency - transaction must leave the database in valid state
  - Isolation - each transaction is independent of other
  - Durability - once transaction is commited ,its chnages are permenant

- ✧ Types of databases :
  - □ Hierarchical databases-tree like structure(IMS DBDC)
  - □ Network databases- graph like structure(IDBMS)
  - □ Object-oriented databases(each info treated as object and accessed through function-objectdb java related)
  - □ Relational databases(each data is related to other somehow-sql,oracle,postgresql)
  - □ Cloud databases(data need to stored and managed virtually-aws,azure,gws)
  - □ Centralised databases(store,located and maintained at single location)
  - □ Nosql databases(key-value pairs, documents,graphs)- used for distributed data(mongodb redis)
  - □ Personal databases
  - □ Operational databases(to delete, update, creating databases)
- ✧ **Database languages:**
  - □ Data definition language(create,alter,drop,truncate,comment,rename)
  - □ Data manipulation language(select,insert,update,delete,merge,explain plan,lock table,call)
  - □ Data control language(grant,revoke)
    - ◆ GRANT privilege_type ON object TO user_or_role;
  - □ Transactional control language(rollback,commit,savepoint)
    - ◆ REVOKE privilege_type ON object FROM user_or_role;
- ✧ **Relational and non-relational databases :**
  - □ Relational(structured schemas , tables)--------->(sql,portgese sql)
  - □ Non-relational(documented, column based,graphs,key-value pair)----->(mongodb)
- ✧ **Relational database - SQL:**
  - Tabular data related to each other (primary key - row , foreign key - tables)
  - SQL DATA TYPE :
    - □ Numeric datatypes(tinyint,smallint,int,bigint,decimal,money,numeric,money,float,real)
    - □ string datatypes(char,varchar,text,nchar,nvarchar,binary(to store media,BLOB),varbinary)
    - □ Date and time datatypes(DATE,TIME,DATETIME,TIMESTAMP)
  - SQL OPERATORS
    - □ Arithmetic operators(+,-,*,/,%)
    - □ Comparison operators(<, >, <=, >=, =, <>)
    - □ Logical operators(AND, OR, NOT)
    - □ Bitwise operators(&, |, ^, ~, <<, >>)
    - □ Compound operators(+=, -=, *=, /=, %=, &=, |=, ^=)

- SPECIAL OPERATORS:
  - ALL(all subqueries should be true)
  - ANY(any of subquery should be true)
  - BETWEEN(used for ranges along with and)
  - IN(to check whether exists in set of values or not)
  - EXISTS(checks the exists of subquery)
  - SOME(used with subqueries)
  - UNIQUE(gives unique rows)
- CREATE DATABASE:
  - Create database database_name/ createdb database_name
  - Show databases ---->displays all dbnames
  - Use database_name ----> to use db
  - Drop database database_name/ destroydb dbname
  - Destroydb IF EXISTS dbname
  - Alter database olddb_name modify name=new name/alterdb olddb_name newname
- CREATE TABLE:
  - Create table table_name(column1 datatype(size), column2 datatype(size)….);
  - Insert into tablename(c1,c2)values(data)
  - Create table newtb_name AS select c1, c2 from oldtb_name where condition;
  - Drop------> deletes the whole rows/table along with its structure(drop table table_name)
  - Truncate -------> delete all rows without removing table structure(truncate table table_name) - cannot be rolled back
  - Delete --------> deletes row by row /deletes all rows once ,can be rolled back to changes(delete from table_name,delete from table tb_name where condition) - can be rolled back
  - Alter table tb_name rename to newtb_name, alter table tb_name rename col_name to newcol_name
    - Alter table tb name add col name datatype
    - Alter table tb name modify column col name datatype
    - Alter table tb name drop column col name
  - Copy/duplicate/backup table ----------> create table table_name as select * from org_tab_name
    create table tab_name as select* from org_tab_name where 1!=1(copies table without data)
  - Local temporary tables ----> create table #tab_name (col1 type….) , insert into #tab_name values(….) , select* from #tab_name
    Drop table #tab_name, automatically drops when connection terminated(which created temp table and same with procedure)
  - Global temporary tables -------->create table ##tab_name
- SQL QUERIES
  - Select top count from tb_name (can use order by , where clause) , select top n percent from tb_name(where , order by)
  - Select * from tb_name where cond limit number , select * from tb_name limit n1 offset n2
  - Select * from tb_name fetch first(select ceil(count(*)/2) from tb_name) rows only (where clause)
  - Select first(col_name) as firstname from tb_name
  - Select last(col name) as last_name from tb_name
  - Select * from tb_name order by rand() limit 1
  - Select * from tb_name where col_name in(v1,v2,v3…) ///// select col_name from tb1 where col_name in (select col_name1 from tb2)
    `EXEC sp_columns tb-name; (to see details of table)`
  - Insert into tb_name1(col_names) select col_names from tb_name2
  - Update tb_name set col_name=v1, c2=v2 where condition
  - Delete from tb_name where condition
  - Select col_names ,count(*) from tab_name grp by col_names having count(*)>1
- SQL CLAUSES
  - Where(between,like,in)--- used to get filtered data
  - With clause (to create temporary relation)
  - Having (group by having order by) -- and/or also used for having
  - Order by (column must be in select statement)- with column name/column number
  - Group by(after where) -- used with aggregate functions(sum,count..)
  - Limit (limit,offset) --- used after order by
- SQL OPERATORS
  - And(gives result which satisfies both conditions) /Or(gives result which satisfies atleast one condition)
  - Like pattern ([aeiou]%)(used with where clause by providing result which matches pattern - case insensitive) , for case sensitive - BINARY
  - In (gives the result which matches the list provided between braces of IN)/ NOT IN ---- used with where clause
  - NOT (used with where --where not condition)
  - NOT EQUAL (returns false ,true and NULL if any exprsn is null) - used with where clause and group by
  - IS NULL (used with where caluse, and,or, count, delete, update)
  - UNION(combines the results of multiple select statements) - each table should have same columns/same datatyeps/same order
    - Provides unique values by default
    - UNION ALL gives result along with duplicate rows
  - EXCEPT(kinda subtract operator, gives result of first select but not second select)
    - Results unique values

        - ◇ To get duplicates use EXCEPT ALL
    - □ BETWEEN (gives result of range where both limits are inclusive)
    - □ ALL (compares every value in subquery/result query)
        - ◇ Used with select,where, grp by, having
        - ◇ Preceded by comparison operators
        - ◇ Uses primary key to comparison
    - □ ANY (compares every value n gives true if atleast on row satisfies condition)
        - ◇ Precede by comparison operators
    - □ DISTINCT (avoids duplicate values)
    - □ INTERSECT (returns only the common rows from both select statement)
        - ◇ Used with where,between/and,like
    - □ EXISTS
    - □ CASE (acts like if else)
        - ◇ Case when cond1 then result1 when cond2 then result2 …… else result end col_name
    - □ Regexp (^word,word$, ".","*","?",[a-z],[0-9])
- SQL AGGREGATE FUNCTIONS
    - □ Count(*) ---- gives total no.of rows(includes null value rows also)
        - ◇ Count(column name) - gives only non null rows
        - ◇ Count(distinct col_name) gives distinct no null rows
    - □ Avg(col_name) , avg(distint col_name)
    - □ Sum(col_name), sum(distinct col_name)
    - □ Min(col_name)
    - □ Max(col_name)
- SQL DATA CONSTRAINTS(for data integrity,reliabiltiy and accuracy)
    - ○ Categories- entity(table),domain(data type),referential(primary and foreign key parent n child dependency) and user defined integrity
    - □ Not NULL - used to specify that column/ value stored should not be null(used with create n alter)
    - □ Unique(no duplicate rows stored in db)
    - □ Primary Key - uniquely identifies the records , not null n unique
        - ◇ Create table
        - ◇ Alter table tb_name add constraint col_name primary name
        - ◇ Simple /composite primary key
    - □ Foreign key -establishes link between two tables
        - ◇ CREATE TABLE child_table (

```
column1 INT PRIMARY KEY,
column2 VARCHAR(50),
parent_id INT,  -- foreign key column
CONSTRAINT fk_parent
  FOREIGN KEY (parent_id)
  REFERENCES parent_table(parent_id)
  ON DELETE CASCADE  -- optional: specifies what happens on delete
  ON UPDATE CASCADE  -- optional: specifies what happens on update
);
```

        - ◇ *ALTER TABLE table_name*
          ADD CONSTRAINT fk_constraint_name FOREIGN KEY (column1, column2, …)
          REFERENCES parent_table(column1, column2, …);
        - ◇ *When record in master table delets n child record exists delete operation fails*
        - ◇ ALTER TABLE table_name DROP CONSTRAINT fk_name;
    - □ Composite key - cannot be null n made by using more than one candidate key
    - □ Alternate key -unique n can be null , more primary keys - one main - rest alternate keys, foreign key is not alternate key
    - □ Check - specified with create table
        - ◇ CREATE TABLE student(
          StudentID INT NOT NULL,
        - ◇ Name VARCHAR(30) NOT NULL,
          Age INT NOT NULL,
          GENDER VARCHAR(9),
          PRIMARY KEY(ID),
          check(Age >= 17)
          );
        - ◇ alter table TABLE_NAME modify COLUMN_NAME check(Predicate);
        - ◇ alter table TABLE_NAME add constraint CHECK_CONST check (Predicate);
        - ◇ alter table TABLE_NAME drop constraint CHECK_CONSTRAINT_NAME;
        - ◇ alter table TABLE_NAME drop check CHECK_CONSTRAINT_NAME;
    - □ Defualt - create column which fills default value if nothing is provided
        - ◇ CREATE TABLE tablename (Columnname ***DEFAULT*** 'defaultvalue');

- ALTER TABLE tablename

  ALTER COLUMN columnname

  DROP DEFAULT;

- SQL JOINS --- where should come after join
  - Join
    - ```
      SELECT s.roll_no, s.name, s.address, s.phone, s.age, sc.course_id
      FROM Student s

      JOIN StudentCourse sc ON s.roll_no = sc.roll_no;
      ```
  - Inner join
    - **SELECT** StudentCourse.COURSE_ID, Student.NAME, Student.AGE **FROM** Student
      INNER JOIN StudentCourse **ON** Student.ROLL_NO = StudentCourse.ROLL_NO;
  - Left join
    - ```
      SELECT table1.column1,table1.column2,table2.column1,....
      FROM table1

      LEFT JOIN table2

      ON table1.matching_column = table2.matching_column;
      ```
  - Right join
    - ```
      SELECT table1.column1,table1.column2,table2.column1,....
      FROM table1

      RIGHT JOIN table2

      ON table1.matching_column = table2.matching_column
      ```
  - Outer join
  - Full join
    - ```
      SELECT table1.column1,table1.column2,table2.column1,....
      FROM table1

      FULL JOIN table2

      ON table1.matching_column = table2.matching_column;
      ```
  - Cross join
  - Self join(manager and employee tables)
  - Update, delete
  - Recursive join
- SQL FUNCTIONS
  - Decimal(10,2)
  - DATE AND TIME FUNCTIONS
    - Now(),curdate(),curtime()
    - Date() - gives only date of date/time expression
    - Extract(unit from date) - extracts each separate data SELECT Name, Extract(DAY FROM BirthTime) AS BirthDay FROM Test;
    - Date_add(date,interval expr type) - to modify the date column /to update
      - SELECT DATE_ADD('2024-11-01', INTERVAL 10 DAY) AS NewDate;
      - Dateadd(interval,number,date)
    - Date_sub() - same syntax as date_add
      - SELECT DATE_SUB('2024-11-01', INTERVAL 10 DAY) AS NewDate;
    - Datediff(date1,date2)
      - SELECT DATEDIFF('2024-11-27', '2024-11-01') AS DateDifference;
    - Date_format/format(date,format specifier) - need to practice (%a,%b,%c……)
  - STRING FUNCTIONS
    - Ascii() , char_length() /len()/length() , concat() ,lower()/lcase() , ucase()/upper(), reverse() , space(digit), strcmp
    - Find_in_set(key,set) , format("0.981","percent") , instr(sentence,alphabet) - occurance of alphabet in sentence
    - Select left/right('sentence',number) - gives left number of digits of sentence
    - Select locate('wordfind','sentence', digit) - gives nth position of word
    - Lpad/rpad('geeks',8,'0') - 000geeks o/p
    - Ltrim/rtrim('123123geeks','123') - geeks o/p-removes leading spaces
    - Mid('geeksforgeeks',6,2) - for o/p - 2 lettrs extracts from 6th position
    - SELECT POSITION('e' IN 'geeksforgeeks') --- 2 o/p -
    - Select repeat('word', no.of times repeated)
    - Substring('sentence', from, size)

- ◇ Substring_index('sentence'','symbol',digit) - gives substring before symbol
- ◇ select TRIM(LEADING '0' FROM '000123'); - 123 o/pn - removes leading and trailing spaces
- ◇ Charindex('letter/substring',string,from position) - gives base address/position of substring
- ◇ REPLACE(string, substring_to_find, substring_to_replace)

□ NUMERIC FUNCTIONS

- ◇ ABS() , acos(number) / asin(number)/atan(number) - gives output in radians
- ◇ Ceil/ceiling - gives smallest integer greater than or equal to given number
- ◇ Cos(angle) , cot(angle) ,sin(),tan()- answer in radians
- ◇ Degrees() /radians()- radians to degrees/vice versa
- ◇ Exp(number) - e raised to power of number
- ◇ Floor(number) - gives largest number less than or equal to given
- ◇ Greatest() , least() , ln() ,log() , log2() , mod(13,2) - gives reminder , pi() , power(4,2) , rand() ,round() , sqrt(),
- ◇ Sign() - 1 for +ve, 0 for -ve
- ◇ Truncate(7.5643,2) - 7.5600 (replaces with 0)

□ STATISTICAL FUNCTIONS

- ◇ Avg(), sum(), count(),max(),min()
- ◇ Var() /variance()- gives population variance
- ◇ Stddev() , standard deviation
- ◇ Percentile_cont(number)
- ◇ Corr(col1,col2)-correlation
- ◇ Covar-pop() - population covariance

□ Working with JSON

- ◇ Bridging for nosql and relational worlds
- ◇ JSON document stored in nvarchar
- ◇ Declare @jsondata nvarchar(max)

```
set @jsondata ={"Information":
{"SchoolDetails":
[{"Name": "VidhyaMandhir"}, {"Name": "Chettinad"}, {"Name":"PSSenior"}]
}
}
```

- ◇ Select isjson(@jsondata) as validjson --- checks whether the givne data is in json format or not
- ◇ Select json_value(@jsondata,'$.information.schooldetails[0].name') as shcoolname --- gives the string stored
- ◇ Select json_query(@jsondata,'.information.schooldetails') as list of schools --- extract the array of data or objects
- ◇ Set @jsondata= json_modify(@jsondata,'$.information.schooldetails[2].name','anjali')
  selectmodifiedjson=@jsondata
- ◇ For json (auto/path) - to export sql data to json document

```
125  SELECT * FROM Authors;
126
127
```

Results Messages

	ID	AuthorName	Age	Skillsets	NumberOfPosts
1	1	Geek	25	Java,Python,.Net	5
2	2	Geek2	22	Android,Python,.Net	15
3	3	Geek3	23	IOS,GO,R	10
4	4	Geek4	24	Java,Python,GO	5

- ◇ Select * from authors for json auto,root(authorinfo) ----- gives o/p as json doc
- ◇ Openjson - to import json to text file

□ CONVERSION FUNCTION

- ◇ Implicit (varcahr/char- number /date ,date/number - varchar/char)
- ◇ Explicit (to_char,to_number,to_date)
- ◇ To_char(number,[format],[nls_parameter]) , to_char(date,'format_model')
- ◇ To_char(number,'format_model') - 9:represent a number,0:forces to give zero,$:places a dollar sign ,".", ","
- ◇ To_date/to_number(char[,'format_model')

□ FUNCTIONS

- ◇ Ltrim(input_string,[trim_characters]) - removes specified chars ,if not provided removes spaces
- ◇ Upper(input_text/column_name)
- ◇ Rtrim(input_text/column_name) - removes right spaces

□ Windows functions

- ◆ Lag,lead used with partition and over

- SQL PARTITIONING

```
CREATE TABLE sales (
  id INT,
  sale_date DATE,
  amount DECIMAL(10, 2)
)
PARTITION BY RANGE (YEAR(sale_date)) (
  PARTITION p2019 VALUES LESS THAN (2020),
  PARTITION p2020 VALUES LESS THAN (2021),
  PARTITION p2021 VALUES LESS THAN (2022)
);

CREATE TABLE employees (
  id INT,
  department VARCHAR(50)
)
PARTITION BY LIST (department) (
```

```
    PARTITION p_sales VALUES IN ('Sales'),
    PARTITION p_marketing VALUES IN ('Marketing'),
    PARTITION p_hr VALUES IN ('HR')
);

CREATE TABLE users (
    id INT,
    username VARCHAR(50)
)
PARTITION BY HASH (id) PARTITIONS 4;
```

- ✧ SQL PROCEDURES - drop procedure if exists procedure_name
  - CREATE PROCEDURE procedure_name (param1 datatype, param2 datatype, ...)

```
[DETERMINISTIC | NOT DETERMINISTIC]
[SQL SECURITY {DEFINER | INVOKER}]
BEGIN
  -- Procedure body (multiple SQL statements)
  statement1;
  statement2;
  ...
END;
```

```
EXAMPLE:-
DELIMITER $$

CREATE PROCEDURE AddNumbers(IN num1 INT, IN num2 INT, OUT result INT)
BEGIN
  SET result = num1 + num2;
END $$

DELIMITER ;
Call/exec addnumber(5,4,@result)
Select @result
```

- ✧ SQL triggers
  - Create trigger schema.trigger_name on table_name after{insert,update,delete}{not ffor replication}as{sql statement}

```
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    stock INT
);

CREATE TABLE order_items (
    order_id INT PRIMARY KEY,
    product_id INT,
    quantity INT,
    FOREIGN KEY (product_id) REFERENCES products (product_id)
);

DELIMITER $$

CREATE TRIGGER update_product_stock
AFTER INSERT ON order_items
FOR EACH ROW
BEGIN
    DECLARE current_stock INT;

    -- Get the current stock of the product
    SELECT stock INTO current_stock
    FROM products
    WHERE product_id = NEW.product_id;

    -- Check if enough stock is available
    IF current_stock < NEW.quantity THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Not enough stock available';
    ELSE
        -- Reduce the stock by the quantity ordered
        UPDATE products
        SET stock = stock - NEW.quantity
        WHERE product_id = NEW.product_id;
    END IF;
END$$

DELIMITER ;
```

- ✧ SQL cursors
  - Employees with a performance score greater than 90 get a bonus of 20% of their salary.
  - Employees with a score between 75 and 90 get a bonus of 10%.
  - Employees with a score below 75 get no bonus.
    - DECLARE

```
  CURSOR employee_cursor IS
    SELECT employee_id, salary, performance_score
    FROM employees
    WHERE performance_score > 70;  -- Only employees with a score > 70
  v_bonus NUMBER;  -- Variable to store the bonus
BEGIN
  FOR employee_record IN employee_cursor LOOP
```

```
        -- Calculate the bonus based on performance score
        IF employee_record.performance_score > 90 THEN
          v_bonus := employee_record.salary * 0.2;
        ELSIF employee_record.performance_score >= 75 THEN
          v_bonus := employee_record.salary * 0.1;
        ELSE
          v_bonus := 0;
        END IF;

        -- Update the employee's bonus in the database
        UPDATE employees
        SET bonus = v_bonus
        WHERE employee_id = employee_record.employee_id;
      END LOOP;

      COMMIT;  -- Commit the changes to the database
    END;
```

✧ Sql functions - drop function if exists function_name

- BASIC SYNTAX -

```
CREATE FUNCTION function_name (parameter1 datatype, parameter2 datatype, ...)
RETURNS return_type
[DETERMINISTIC | NOT DETERMINISTIC]
[SQL SECURITY {INVOKER | DEFINER}]
[characteristics]
BEGIN
  -- function body
  RETURN expression;
END;
```

-
```
CREATE FUNCTION dbo.fn_AddNumbers (@num1 INT, @num2 INT)
RETURNS INT
AS
BEGIN
  RETURN @num1 + @num2
END; - to fetch select dbo.fn_AddNumbers(5,10)
```

-
```
CREATE FUNCTION dbo.fn_GetEmployeesByDept (@DepartmentID INT)
RETURNS TABLE
AS
RETURN
(
  SELECT EmployeeID, EmployeeName
  FROM Employees
  WHERE DepartmentID = @DepartmentID
); - to fetch select dbo.fn_getemployeesbydept(1)
```

-
```
DELIMITER //
CREATE FUNCTION AddNumbers(num1 INT, num2 INT)
RETURNS INT
DETERMINISTIC
BEGIN
  RETURN num1 + num2;
END //
DELIMITER ;- to fetch select addnumbers(5,10)
```

Learnings during practice

20 October 2024 19:01

- Lag,lead() over()
- Group_concat
- LEAD(expression, offset, default) OVER (PARTITION BY partition_expression ORDER BY order_expression)