DBMS

Week 03

Chapter 3 Introduction to SQL

Structured Query Language (SQL) is a powerful and standardized programming language used for managing and manipulating relational databases. SQL enables users to interact with databases by defining and executing various operations, such as querying data, inserting new records, updating existing records, and more. It provides a common interface for users to communicate with the database management system (DBMS) and retrieve or modify data without needing to understand the underlying details of data storage.

Overview of the SQL Query Language

Origins and Evolution of SQL

- IBM developed the original version of SQL, initially known as Sequel, as part of the System R project in the early 1970s.
- Sequel evolved over time and was eventually renamed SQL, becoming the standard language for relational databases.
- SQL is supported by numerous database products and has become the established standard for interacting with relational databases.

SQL Standardization

- The American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) published the SQL-86 standard in 1986.
- ANSI published an extended standard called SQL-89 in 1989.
- Subsequent versions of the SQL standard include SQL-92, SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011, and SQL:2016.

Components of SQL

- Data-Definition Language (DDL): In SQL, DDL commands are used to define relation schemas, modify them, and manage relations.
- Data-Manipulation Language (DML): SQL's DML allows querying data from the database and performing operations like inserting, deleting, and modifying tuples.
- Integrity: SQL DDL includes commands to enforce integrity constraints on the data stored in the database, ensuring data validity.
- View Definition: SQL supports creating views, which are virtual tables defined using queries.
- Transaction Control: SQL provides commands to manage the initiation and completion of transactions.
- Embedded SQL and Dynamic SQL: Embedded and dynamic SQL explain how SQL statements can be included within general-purpose programming languages.
- Authorization: SQL DDL includes commands to specify access rights to relations and views.

SQL Data Definition

Data-Definition Language (DDL)

- DDL in SQL is used to specify the structure and characteristics of relations (tables) in a database.
- It includes information about the schema, data types, integrity constraints, indices, security, authorization, and storage structure of relations.

Basic Data Types

- char(n) or character(n): Fixed-length character string with length n.
- varchar(n) or character varying(n): Variable-length character string with maximum length n.
- int or integer: Integer (machine-dependent subset).
- smallint: Small integer (machine-dependent subset of integer).
- numeric(p, d): Fixed-point number with precision p and d digits after the decimal point.
- real and double precision: Floating-point and double-precision floating-point numbers.
- float(n): Floating-point number with precision of at least n digits.

Basic Schema Definition

- An SQL relation (table) is created using the create table command.
- The command specifies relation name, attributes, and their data types.
- Attributes can also include optional constraints.
- Example:
  - create table department
  - (dept_name varchar(20),
  - building varchar(15),
  - budget numeric(12,2),
  - primary key (dept_name));
- The primary key constraint enforces non-null and uniqueness for the specified attributes.
- Additional integrity constraints include foreign key and not null.

Foreign Key Constraints

- Foreign key constraints establish relationships between tables.
- They ensure that values in certain attributes of one table correspond to the primary key values in another table.
- Example: Ensuring dept_name in the course table references the primary key in the department table.

Null Constraint

- not null constraint prevents attributes from having null values.
- For instance, the name attribute of the instructor relation cannot have null values.

Modifying Relations

- alter table command is used to modify existing relations.
- You can add attributes using add and drop attributes using drop.

Dropping Relations

- Use the drop table command to remove a relation from the database.
- This removes both the schema and the data, unlike just deleting tuples with delete.

Basic Structure of SQL Queries

Structure of an SQL Query

- An SQL query consists of three main clauses: select, from, and where.
- The select clause specifies the attributes to be included in the result.
- The from clause lists the relations (tables) involved in the query.
- The where clause includes a predicate that filters the rows from the relations.

Queries on a Single Relation

- Queries on a single relation retrieve information from that relation.
- Examples include:
  - Finding specific attributes: select name from instructor;
  - Eliminating duplicates: select distinct dept name from instructor;
  - Performing arithmetic operations in the select clause.

Queries on Multiple Relations

- Queries often involve multiple relations, requiring joining and filtering.
- Cartesian product: The result of the from clause is a Cartesian product of the relations listed.
- The where clause restricts combinations based on a predicate.
- An example query combines instructor and teaches relations: select name, course id from instructor, teaches where instructor.ID= teaches.ID;

General Query Structure

- An SQL query follows this order of execution:
  - Generate Cartesian product of relations from the from clause.
  - Apply predicates from the where clause.
  - Select attributes as specified in the select clause.

Avoiding Cartesian Product Pitfalls

- Careful use of predicates in the where clause is essential to avoid generating large Cartesian products.
- Omitting the where clause can lead to unintended huge result sets.
- Real implementations of SQL optimize query execution, generating only necessary elements of the Cartesian product.

Predicate Logic and Expressions

- The where clause uses predicate logic with comparison operators (<, >, =, etc.).
- Logical operators (and, or, not) allow combining conditions.
- Expressions can involve arithmetic operations and comparisons.

Additional Basic Operations

The Rename Operation

- SQL allows renaming attributes and relations using the as clause.
- It can appear in both the select and from clauses.
- Used to change attribute names in the result or to assign an alias to relations.
- Helps avoid naming conflicts, enables clearer code, and supports self-joins.
- Example: select name as instructor name, course id from instructor, teaches where instructor.ID= teaches.ID;

String Operations

- SQL handles strings enclosed in single quotes ('string').
- Escaping single quotes: Use two consecutive single quotes ('').
- SQL offers various string functions: concatenation (using ||), substring extraction, length calculation, case conversion, and more.
- Pattern matching using the like operator with % and _ wildcards.
- Escape characters can be specified using the escape keyword.
- Example: select dept name from department where building like '%Watson%';

Attribute Specification in the Select Clause

- The asterisk * can be used to select all attributes in the select clause.
- Example: select instructor.* from instructor, teaches where instructor.ID= teaches.ID;

Ordering the Display of Tuples

- The order by clause sorts the result tuples in ascending order by default.
- Use desc for descending order and asc for ascending order.
- Sorting can be based on multiple attributes.
- Example: select * from instructor order by salary desc, name asc;

Where-Clause Predicates

- SQL offers the between comparison operator to simplify range conditions.
- Example: select name from instructor where salary between 90000 and 100000;
- not between can be used for negation.
- Tuples can be compared using the lexicographic ordering of their attributes.
- Example: select name, course id from instructor, teaches where (instructor.ID, dept name) = (teaches.ID, 'Biology');

Set Operations

UNION Operation

- The UNION operation combines the result sets of two queries, removing duplicates by default.
- Syntax: (query1) UNION (query2);
- Example:
  - (select course id from section where semester = 'Fall' and year = 2017) UNION (select course id from section where semester = 'Spring' and year = 2018);
- The result contains a unique list of courses taught in either Fall 2017 or Spring 2018 or both, eliminating duplicates.

INTERSECT Operation

- The INTERSECT operation returns the common elements of two queries' result sets, removing duplicates.
- Syntax: (query1) INTERSECT (query2);
- Example:
  - (select course id from section where semester = 'Fall' and year = 2017) INTERSECT (select course id from section where semester = 'Spring' and year = 2018);
- The result contains courses taught in both Fall 2017 and Spring 2018, eliminating duplicates.

EXCEPT Operation

- The EXCEPT operation returns the elements present in the first query but not in the second query, removing duplicates.
- Syntax: (query1) EXCEPT (query2);
- Example:
  - (select course id from section where semester = 'Fall' and year = 2017) EXCEPT (select course id from section where semester = 'Spring' and year = 2018);
- The result contains courses taught in Fall 2017 but not in Spring 2018, eliminating duplicates.
- Note:
  - If you want to retain duplicate values in the result, you can use UNION ALL, INTERSECT ALL, or EXCEPT ALL. Some database systems might use different syntax or keywords for these operations, such as MINUS instead of EXCEPT. The operations automatically handle duplicates by default, ensuring that only unique tuples are present in the result.

Null Values

Arithmetic Expressions with NULL

- When performing arithmetic operations involving NULL values, the result is always NULL.
- For example, if you have an expression like A + 5, and the value of A is NULL, then the result of the expression will also be NULL.

Comparison Operations with NULL

- Comparisons involving NULL values result in an "unknown" outcome, rather than a true or false outcome.
- For instance, the comparison 1 < NULL is not true, but it's also not false; it's unknown.
- SQL treats comparisons with NULL values as producing an unknown result because the actual value of the NULL is unknown.

Logical Operations with NULL

- Logical operations (AND, OR, NOT) involving NULL values are also extended to handle unknown outcomes.
- true AND unknown results in unknown, false AND unknown results in false, and unknown AND unknown results in unknown.
- true OR unknown results in true, false OR unknown results in unknown, and unknown OR unknown results in unknown.
- NOT unknown results in unknown.

Handling NULLs in WHERE Clauses

- In WHERE clauses, if a predicate evaluates to either false or unknown for a tuple, that tuple is not included in the result.
- You can use the IS NULL predicate to check for NULL values in a column, and IS NOT NULL to check for non-NULL values.

Comparing with Unknown and NULL

- SQL allows you to explicitly check if a comparison result is unknown using the IS UNKNOWN and IS NOT UNKNOWN clauses.
- For example, salary > 10000 IS UNKNOWN will check if the comparison result is unknown.

Using DISTINCT with NULLs

- When using the SELECT DISTINCT clause, NULL values are treated as equal for the purpose of eliminating duplicates.
- This means that two tuples with the same attribute values (including NULLs) will be treated as identical when using DISTINCT.

Set Operations with NULLs

- The set operations UNION, INTERSECT, and EXCEPT treat tuples as identical if their values for all attributes are the same, even if some of those values are NULL.

Aggregate Functions

Aggregate Functions in SQL

- SQL provides several built-in aggregate functions: avg, min, max, sum, and count.
- These functions operate on a collection of values (a set or multiset) and return a single value.

Types of Aggregate Functions

- avg: Calculates the average of a set of numeric values.
- min: Returns the minimum value from a set of values.
- max: Returns the maximum value from a set of values.
- sum: Computes the sum of a set of numeric values.
- count: Counts the number of items in a set.

Using Aggregate Functions

- Aggregate functions are typically used in the SELECT clause of SQL queries.
- The GROUP BY clause is used to group data before applying aggregate functions.
- The HAVING clause allows you to filter the grouped data after applying aggregate functions.

Handling Null Values and Aggregation

- Aggregate functions treat null values differently depending on the function:
  - avg and sum: Ignore null values.
  - min and max: Include null values.
  - count: Includes null values unless COUNT(*) is used.

Aggregation with Boolean Values

- SQL introduced a Boolean data type that can have values: true, false, and unknown.
- The aggregate functions some (equivalent to OR) and every (equivalent to AND) work with Boolean values in a collection.

Handling Aggregation with Grouping

- The GROUP BY clause is used to group rows based on specific attributes.
- Aggregate functions can be applied to each group.
- The HAVING clause can be used to filter groups based on aggregate results.

Naming Aggregate Results

- Aggregate results can be given meaningful names using the AS clause.

Handling Duplicate Elimination

- The DISTINCT keyword can be used with aggregate functions to eliminate duplicates before aggregation.

Usage of Nulls and Aggregation

- The effect of null values on aggregate functions varies, with some functions ignoring nulls and others including them in calculations.