mood-book

UNIT-3
SQL: QUERIES, CONSTRAINTS, TRIGGERS

The Form of a Basic SQL Query:

SELECT [DISTINCT] select-list
FROM from-list
WHERE qualification

Every query must have a SELECT clause, which specifies columns to be retained in the result, and a FROM clause, which specifies a cross-product of tables. The optional WHERE clause specifies selection conditions on the tables mentioned in the FROM clause.

Eg: 1. Find the names and ages of all sailors.

```
SELECT DISTINCT S.sname, S.age
FROM Sailors S
```

2.Find all sailors with a rating above 7.

```
SELECT S.sid, S.sname, S.rating, S.age
FROM Sailors AS S
WHERE S.rating > 7
```

We now consider the syntax of a basic SQL query in detail.

- The from-list in the FROM clause is a list of table names. A table name can be followed by a range variable; a range variable is particularly useful when the same table name appears more than once in the from-list.
- The select-list is a list of (expressions involving) column names of tables named in the from-list. Column names can be prefixed by a range variable.
- The qualification in the WHERE clause is a boolean combination (i.e., an expression using the logical connectives AND, OR, and NOT) of conditions of the form expression op expression, where op is one of the comparison operators {<, <=, = , <>, >=,>}.An expression is a column name, a constant, or an (arithmetic or string) expression.
- The DISTINCT keyword is optional. It indicates that the table computed as an answer to this query should not contain duplicates, that is, two copies of the same row. The default is that duplicates are not eliminated.

The following is the conceptual evaluation strategy of SQL query

1. Compute the cross-product of the tables in the from-list.

2. Delete those rows in the cross-product that fail the qualification conditions.

3. Delete all columns that do not appear in the select-list.

4. If DISTINCT is specified, eliminate duplicate rows.

UNION, INTERSECT, AND EXCEPT:

SQL provides three set-manipulation constructs that extend the basic query form. Since the answer to a query is a multiset of rows, it is natural to consider the use of operations such as union, intersection, and difference. SQL supports these operations under the names UNION, INTERSECT, and EXCEPT.

UNION:

- The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
- The union operation eliminates the duplicate rows from its result set.

Eg: Find the names of sailors who have reserved a red or a green boat.

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = ‘red’
UNION
SELECT S2.sname
FROM Sailors S2, Boats B2, Reserves R2
WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = ‘green’
```

This query says that we want the union of the set of sailors who have reserved red boats and the set of sailors who have reserved green boats.

INTERSECT:

- The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of datatype and columns must be the same.
- It has no duplicates and it arranges the data in ascending order by default.

Eg: Find the names of sailors who have reserved both a red and a green boat.

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = ‘red’
INTERSECT
SELECT S2.sname
FROM Sailors S2, Boats B2, Reserves R2
WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = ‘green’
```

EXCEPT:

- Except operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.

Eg: Find the sids of all sailors who have reserved red boats but not green boats.

```
SELECT S.sid
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
EXCEPT
SELECT S2.sid
FROM Sailors S2, Reserves R2, Boats B2
WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'
```

SQL also provides other set operations: IN (to check if an element is in a given set), op ANY, op ALL (to compare a value with the elements in a given set, using comparison operator op), and EXISTS (to check if a set is empty). IN and EXISTS can be prefixed by NOT, with the obvious modification to their meaning.

Nested Queries:

A nested query is a query that has another query embedded within it; the embedded query is called a subquery.

SQL provides other set operations: IN (to check if an element is in a given set),NOT IN(to check if an element is not in a given set).

Eg:1. Find the names of sailors who have reserved boat 103.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN ( SELECT R.sid
                 FROM Reserves R
                 WHERE R.bid = 103 )
```

The nested subquery computes the (multi)set of sids for sailors who have reserved boat 103, and the top- level query retrieves the names of sailors whose sid is in this set. The IN operator allows us to test whether a value is in a given set of elements; an SQL query is used to generate the set to be tested.

2.Find the names of sailors who have not reserved a red boat.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid NOT IN ( SELECT R.sid
                     FROM Reserves R
                     WHERE R.bid IN ( SELECT B.bid
                                      FROM Boats B
                                      WHERE B.color = 'red' )
```

Correlated Nested Queries:

In the nested queries that we have seen, the inner subquery has been completely independent of the outer query. In general the inner subquery could depend on the row that is currently being examined in the outer query.

Eg: Find the names of sailors who have reserved boat number 103.

```
SELECT S.sname
FROM Sailors S
WHERE EXISTS ( SELECT *
               FROM Reserves R
               WHERE R.bid = 103 AND R.sid = S.sid )
```

The EXISTS operator is another set comparison operator, such as IN. It allows us to test whether a set is nonempty.

Set-Comparison Operators:

SQL also supports op ANY and op ALL, where op is one of the arithmetic comparison operators {<, <=,
=, <>, >=,>}.

Eg:1. Find sailors whose rating is better than some sailor called Horatio.

```
SELECT S.sid FROM Sailors S
WHERE S.rating > ANY ( SELECT S2.rating
FROM Sailors S2
WHERE S2.sname = 'Horatio' )
```

If there are several sailors called Horatio, this query finds all sailors whose rating is better than that of some sailor called Horatio.

2.Find the sailors with the highest rating.

```
SELECT S.sid FROM Sailors S
WHERE S.rating >= ALL ( SELECT S2.rating
FROM Sailors S2 )
```

Aggregate Operators:

SQL supports five aggregate operations, which can be applied on any column

1. COUNT ([DISTINCT] A): The number of (unique) values in the A column.

2. SUM ([DISTINCT] A): The sum of all (unique) values in the A column.

3. AVG ([DISTINCT] A): The average of all (unique) values in the A column.

4. MAX (A): The maximum value in the A column.

5. MIN (A): The minimum value in the A column.

Eg:1. Find the average age of all sailors.

```
SELECT AVG (S.age)
FROM Sailors S
```

2.Count the number of sailors.

```
SELECT COUNT (*)
FROM Sailors S
```

The GROUP BY and HAVING Clauses

Thus far, we have applied aggregate operations to all (qualifying) rows in a relation. Often we want to apply aggregate operations to each of a number of groups of rows in a relation, where the number of groups depends on the relation instance.
Syntax:

```
SELECT [ DISTINCT ] select-list
FROM from-list
WHERE qualification
GROUP BY grouping-list
HAVING group-qualification
```

Eg:Find the age of the youngest sailor for each rating level.

```
SELECT S.rating, MIN (S.age)
FROM Sailors S
GROUP BY S.rating
```

NULL Values:

SQL provides a special column value called null to use where some column does not have a value to hold or the value is unknown. We use null when the column value is either unknown or inapplicable.

Logical Connectives AND, OR, and NOT

The logical operators AND, OR, and NOT using a three-valued logic in which expressions evaluate to true, false, or unknown. OR of two arguments evaluates to true if either argument evaluates to true, and to unknown if one argument evaluates to false and the other evaluates to unknown. (If both arguments are false, of course, it evaluates to false.) AND of two arguments evaluates to false if either argument evaluates to false, and to unknown if one argument evaluates to unknown and the other evaluates to true or unknown.

Outer Joins

The join operation that rely on null values, called outer joins, are supported in SQL. Consider the join of two tables, say Sailors & Reserves. Tuples of Sailors that do not match some row in Reserves according to the join condition c do not appear in the result. In an outer join, on the other hand, Sailor rows without a matching Reserves row appear exactly once in the result, with the result columns inherited from Reserves assigned null values.

In fact, there are several variants of the outer join idea. In a left outer join, Sailor rows without a matching Reserves row appear in the result, but not vice versa. In a right outer join, Reserves rows without a matching Sailors row appear in the result, but not vice versa. In a full outer join, both Sailors and Reserves rows without a match appear in the result.

Triggers and Active Databases:

A trigger is a procedure that is automatically invoked by the DBMS in response to specified changes to the database, and is typically specified by the DBA. A database that has a set of associated triggers is called an active database.

A trigger description contains three parts:

- Event: A change to the database that activates the trigger.
- Condition: A query or test that is run when the trigger is activated.
- Action: A procedure that is executed when the trigger is activated and its condition is true.

Eg: The trigger called init count initializes a counter variable before every execution of an INSERT statement that adds tuples to the Students relation. The trigger called incr count increments the counter for each inserted tuple that satisfies the condition age < 18.

```
CREATE TRIGGER init_count BEFORE INSERT ON Students        /* Event */
    DECLARE
        count INTEGER;
    BEGIN                                                  /* Action */
        count := 0;
    END

CREATE TRIGGER incr_count AFTER INSERT ON Students         /* Event */
    WHEN (new.age < 18)                                    /* Condition*/
    FOR EACH ROW
    BEGIN                                                  /* Action */
        count:=count+1;
    END
```