Databases Design. Introduction to SQL

LECTURE 9

Queries

SELECT statement

- Query operations facilitate data retrieval from one or more tables.
- The result of any query is a table. The result can be further manipulated by other query operations.
- Syntax:

```
SELECT attribute(s)
FROM table(s)
[WHERE selection condition(s)];
```

Aliasing in SQL

- A PostgreSQL alias assigns a table or a column a temporary name in a query. The aliases only exist during the execution of the query.
- The following illustrates the syntax of the table alias:

```
SELECT column_list
FROM table_name AS alias_name;
```

- The AS keyword in the table alias syntax is optional.

Aliasing in SQL

The table alias has several uses:

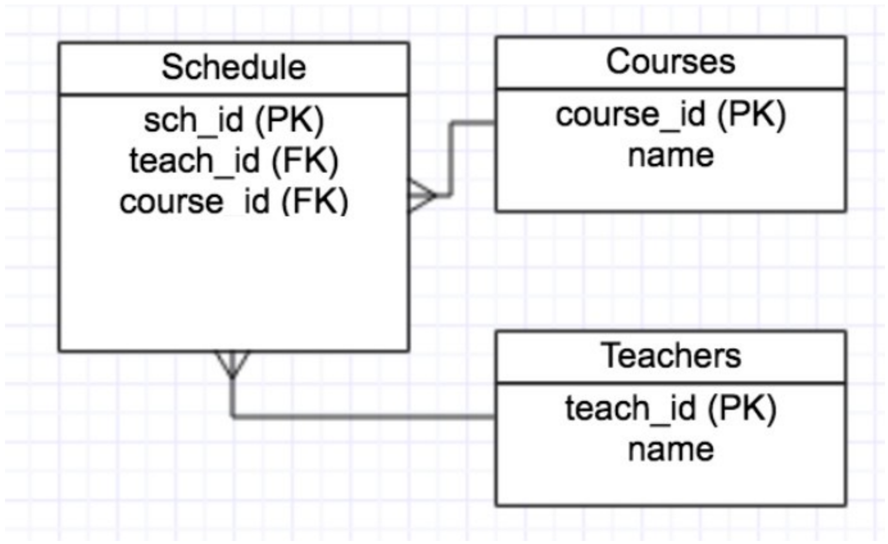
- First, if you must qualify a column name with a long table name, you can use the table alias to make your query more readable.

- The practical uses are when you query data from multiple tables that have the same column names. In this case, you must qualify the columns using the table names.

Aliasing in SQL

- Aliasing table names during join operations makes them a lot more understandable.

```
SELECT c.name, t.name
FROM Courses c, Teachers t, Schedule s
WHERE c.course_id = s.course_id AND
t.teach_id = s.teach_id;
```



Aliasing in SQL

- The following shows the syntax of column alias:

```
SELECT column_name AS alias_name
 FROM table_name;
```

  - In this syntax, the column_name is assigned as alias_name. The AS keyword is optional

- Rename the fname column to First_Name:

```
SELECT fname AS First_name
 FROM Students;
```

String Concatenation

- In the Students table first and last names are stored as two attributes. For combining them into one column, use the || operator:

  ```
  SELECT fname || lname
  FROM Students;
  ```

- Notice that the names concatenated together without a space in between. We can add such a space using:

  ```
  SELECT fname || ‘ ‘ || lname
  FROM Students;
  ```

Distinct Results

- The DISTINCT clause is used in the SELECT statement to remove duplicate rows from a result set.
  The DISTINCT clause keeps one row for each group of duplicates.
- The syntax of the DISTINCT clause:

```
SELECT DISTINCT column_name
FROM table_name;
```

- To select the distinct last names from the Students:

```
SELECT DISTINCT lname
 FROM Students;
```

Distinct Results

If you specify multiple columns, the DISTINCT clause will evaluate the duplicate based on the combination of values of these columns.

```
SELECT DISTINCT column_1, column_2
FROM table_name;
```

In this case, the combination of both column_1 and column_2 will be used for evaluating duplicate.

NULL Values

- NULL indicates absence of a value in a column.
- NULL is not a value, therefore, you cannot compare it with any value like a number or a string.
- Since NULL may appear in a column, we must be able to detect its presence.
- For this reason, SQL provides the IS NULL and IS NOT NULL operators.

NULL Values

- Consider the following query:

```
SELECT stud_id, fname
FROM Students
WHERE group_id IS NULL;
```

- This query returns record of each student where the group_id is null (is empty).

IS NULL and IS NOT NULL

Students table in the database

stud_id	fname	group_id
1	student1	2
2	student2	2
3	student3	

… WHERE group_id IS NULL;

stud_id	fname
3	student3

… WHERE group_id IS NOT NULL;

stud_id	fname
1	student1
2	student2

Comparison Operators

- One of the most common selection conditions is a range condition. Range condition filters results where the values in a column are between one or two values.

- There are two ways to perform a range operation:
  - Using the <, <=, >, >= operators.
  - Using the BETWEEN operator.

Comparison Operators

Operator	**Description**
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
=	equal
<> or !=	not equal

- Comparison operators are available for all relevant data types.
- All comparison operators are binary operators that return values of type Boolean.
- expressions like 1 < 2 < 3 are not valid (because there is no < operator to compare a Boolean value with 3).

Comparison Operators

- A range condition is specified using the <,<=,> and >= operators as
  ```
  SELECT …
  FROM …
  WHERE column < value1 AND column > value2;
  ```

- Example: Query the first and last names of all students with GPA between 3 and 4:
  ```
  SELECT fname, lname
  FROM Students
  WHERE gpa >= 3 AND gpa <= 4;
  ```

BETWEEN operator

- We may render the same select condition in a form that is closer to English using the BETWEEN operator.

- The query on the previous slide can be rewritten as

```
SELECT fname, lname
FROM Students
WHERE gpa BETWEEN 3 AND 4;
```

Comparison Operators

- The BETWEEN operator has a negation: NOT BETWEEN.
- The BETWEEN operator is defined for most data types including numeric and temporal data.

BETWEEN and NOT BETWEEN

BETWEEN treats the endpoint values as included in the range. NOT BETWEEN does the opposite comparison.

a BETWEEN x AND y

is equivalent to

a >= x AND a <= y

a NOT BETWEEN x AND y

is equivalent to

a < x OR a > y

Pattern Matching

SQL provides the

- LIKE operator to support comparisons of partial strings;
- *%* and _ characters to match strings.

The LIKE operator is used in conjunction with *%* and _ characters.

Pattern Matching

- The % character matches an arbitrary number of characters, including spaces.
- So, vinc% would match each of the following:

  vince, vincent, vincenzo, vinc

- The _ character matches a single arbitrary character.
- So, v_nce will match each of the following:

  vince, vance, vbnce, vnnce, v1nce, and so on.

Pattern Matching

- Example with %: Query the phone number if it starts with 412.

```
SELECT phone
FROM Students
WHERE phone LIKE ‘412%’;
```

Pattern Matching

- Example with _: Query the phone number if it starts with ‘20’ and ends with ‘-555-4335’.

```
SELECT phone
FROM Students
WHERE phone LIKE ‘20_-555-4335’;
```

Converting Data Types

- PostgreSQL CAST is used to convert from one data type into another.
- First, you specify an expression that can be a constant or a table column, that you want to convert. Then, you specify the target type which you want to convert to.
- Syntax:

```
CAST (expression AS type)
```

- Example:

```
SELECT CAST ('100' AS INTEGER);
SELECT CAST (phone AS varchar (20))
FROM Students;
```

Converting Data Types

- Besides the type CAST syntax, following syntax can be used to convert a type into another:

  expression::type

- Notice that the cast syntax with :: is PostgreSQL specific and does not conform to SQL.
- Example:

  SELECT '100'::INTEGER;

Books

Connolly, Thomas M. Database Systems: A Practical Approach to Design, Implementation, and Management / Thomas M. Connolly, Carolyn E. Begg.- United States of America: Pearson Education

Garcia-Molina, H. Database system: The Complete Book / Hector Garcia-Molina.- United States of America: Pearson Prentice Hall

Sharma, N. Database Fundamentals: A book for the community by the community / Neeraj Sharma, Liviu Perniu.- Canada

www.postgresql.org/docs/manuals/

www.postgresql.org/docs/books/

Online SQL Training

- sqlzoo.net
- sql-ex.ru