Databases Design. Introduction to SQL

LECTURE 10

Queries

Last lecture

- AS
- String Concatenation ||
- DISTINCT
- IS NULL & IS NOT NULL
- Range condition:

  <,<=,>, >=

  BETWEEN & NOT BETWEEN
- LIKE with % and _ characters
- CAST & ::

Aggregate Functions

SQL provides the following aggregate functions that appear in SELECT statement:

- Min() selects the minimum value
- Max() selects the maximum value
- Avg() selects the average value
- Sum() selects the sum of occurrences
- Count() selects the number of occurrences

SQL aggregate functions return a single value, calculated from values in a column.

Aggregate Functions

- Example: Select the minimum, maximum and average gpa from the Students table.

```
SELECT min(gpa), max(gpa), avg(gpa)
FROM Students;
```

Aggregate Functions

- Selecting count(*) or count(expression) returns the number of tuples that satisfy a selection condition.

- Example: Get number of students.

```
SELECT count(*)
FROM Students;
```

Aggregate Functions

- Example: Get number of students with group_id = 1. The column should be named NumOfStud.

```
SELECT count(*) AS NumOfStud
FROM Students
WHERE group_id=1;
```

Count example

Students table

stud_id	fname	group_id
1	student1	2
2	student2	2
3	student3	

Count (*)

count
3

Count (group_id)

count
2

GROUP BY

- The aggregate functions can also be applied to subsets of tables.
- In SQL, rows can be grouped together based on the value of some attribute(s) called **grouping attribute**.
- The GROUP BY clause is used to specify these groupings.
- The effect is to combine each set of rows having common values into one group row that represents all rows in the group. This is done to compute aggregates that apply to these groups.

GROUP BY: example

- Example: Select the group_id’s that students study in and the number of students that study in those groups.

  ```
  SELECT group_id, count(*)
  FROM Students
  GROUP BY group_id;
  ```

- Note: The group by attribute (group_id) should be part of the selected columns.

GROUP BY: example

Students table

stud_id	fname	group_id
1	student1	1
2	student2	1
3	student3	2

```
SELECT count(*)
FROM Students;
```

count
3

GROUP BY: example

Students table

stud_id	fname	group_id
1	student1	1
2	student2	1
3	student3	2

```
SELECT group_id, count(*)
FROM Students
GROUP BY group_id;
```

group_id	count
1	2
2	1

HAVING

- The HAVING clause is used to filtering groups

- This is similar to a selection condition WHERE only the required rows are filtered out

HAVING: example

- Query the group_id's and number of students of each group.

  ```
  SELECT group_id, count(*)
  FROM Students
  GROUP BY group_id;
  ```

- Now, query group_id's with more than 20 students.

  ```
  SELECT group_id, count(*)
  FROM Students
  GROUP BY group_id
  HAVING count(*) > 20;
  ```

Example with join

```
SELECT g.name as group_name,
count(*) as number_of_students
FROM Students s, Groups g
WHERE s.group_id=g.group_id
GROUP BY g.name
HAVING count(*) > 20;
```

group_name	number_of_students
CSSE-131	21
CSSE-132	24
…	…

ORDER BY

- The ORDER BY clause is used to set the ordering of the resultant table.
- Columns may be sorted in ASCending or DESCending order.

- Example: Query the first and last name of each student in ascending order of their last names

  ```
  SELECT fname, lname
  FROM  Students
  ORDER BY lname ASC;
  ```

Ordering Results in SQL: example

- The ordering of results may be mixed in query: one column may be sorted in ascending order while another column may be sorted in descending order.
- For the previous query, sort the results in ascending order of last names and descending order of first names:

```
SELECT fname, lname
FROM Students
ORDER BY lname ASC, fname DESC;
```

Example with join

```
SELECT g.name as group_name, count(*)
as number_of_students
FROM Students s, Groups g
WHERE s.group_id=g.group_id
GROUP BY g.name
HAVING count(*) > 20
ORDER BY g.name ASC;
```

group_name	number_of_students
CSSE-131	21
CSSE-132	24
…	…

SELECT Statement

- SQL allows us to query data using *select* statements. Syntax:

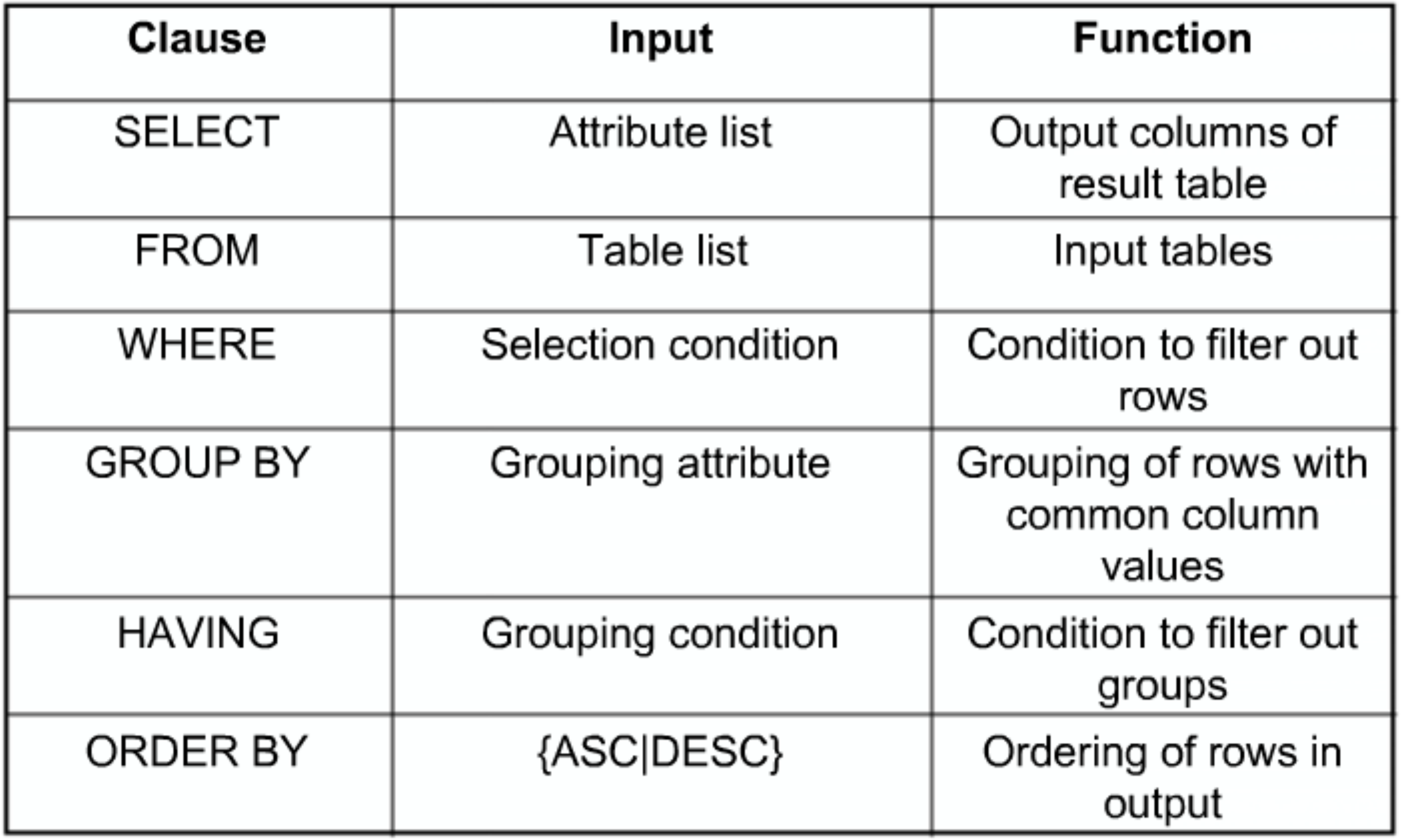
```
SELECT attribute(s)
FROM table(s)
WHERE selection condition(s);
```

Complete SELECT Statement

```
SELECT attribute(s)
FROM table(s)

[WHERE selection condition(s)]
[GROUP BY condition(s)]
[HAVING  condition(s)]
[ORDER BY condition(s)]
```

Select Statement Summary



Clause	Input	Function
SELECT	Attribute list	Output columns of result table
FROM	Table list	Input tables
WHERE	Selection condition	Condition to filter out rows
GROUP BY	Grouping attribute	Grouping of rows with common column values
HAVING	Grouping condition	Condition to filter out groups
ORDER BY	{ASC|DESC}	Ordering of rows in output

String Functions and Operators

Function	Description	Example	Result
bit_length(string)	Number of bits in string	bit_length('jose')	32
length(string) or char_length(string)	Number of characters in string	length('jose')	4
lower(string)	Convert string to lower case	lower('TOM')	tom
upper(string)	Convert string to upper case	upper('tom')	TOM
substring(string [from int] [for int])	Extract substring	substring('Thomas' from 2 for 3)	hom

String Functions and Operators

Function	Description	Example	Result
left(str text, n int)	Return first n characters in the string. When n is negative, return all but last |n| characters.	left('abcde', 2)	ab
right(str text, n int)	Return last n characters in the string. When n is negative, return all but first |n| characters.	right('abcde', 2)	de
replace(string text, fromtext, to text)	Replace all occurrences in string of substring from with substring to	replace('abcdefabcdef', 'cd', 'XX')	abXXefabXXef
reverse(str)	Return reversed string	reverse('abcde')	edcba

Date Functions

EXTRACT (*field* FROM *source*)

EXTRACT function retrieves subfiels such as year or hour from date/time values.

Source must be a value expression of date type.

Field is an identifier or string that selects what field to extract from the source value.

Date Functions

date_part (‘*field*’, *source*)

Source must be a value expression of date type.

Field is an identifier or string that selects what field to extract from the source value.

Date Functions

***Field*s:**

- century
- year
- month
- week
- day
- decade
- quarter
- dow (the day of the week) / isodow
- doy (day of the the year)
- hour
- minute
- second
- etc.

EXTRACT / date_part examples

```
SELECT EXTRACT(year FROM bdate)
FROM Students;
```

```
SELECT date_part('year', bdate)
FROM Students;
```

Date Functions

```
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
```

Example:

```
SELECT CURRENT_DATE;
```

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/