Все словечки из лаб по базам данных, которые пригодятся (для начала)

1. INSERT

* **Purpose:** Adds new rows to a table.
* **Example:**
```
INSERT INTO students (name, age) VALUES ('John Doe', 21);
```
Comment: Adds a new student named 'John Doe' with age 21 to the 'students' table.

2. INTO

* **Purpose:** Specifies the table where new rows will be inserted or existing rows will be updated.
* **Example:**
```
INSERT INTO students (name, age) VALUES ('John Doe', 21);
```
Comment: Inserts new rows into the 'students' table.

3. SELECT

* **Purpose:** Retrieves data from one or more tables.
* **Example:**
```
SELECT * FROM students WHERE name = 'John Doe';
```
Comment: Selects all columns for students named 'John Doe' from the 'students' table.

4. UPDATE

* **Purpose:** Modifies existing rows in a table.

* **Example:**
```
UPDATE students SET age = 22 WHERE name = 'John Doe';
```
Comment: Updates the age of 'John Doe' to 22 in the 'students' table.

5. DELETE

* **Purpose:** Removes rows from a table.
* **Example:**
```
DELETE FROM students WHERE name = 'John Doe';
```
Comment: Deletes the row for 'John Doe' from the 'students' table.

6. ALTER

* **Purpose:** Modifies the structure of a table, such as adding or removing columns.
* **Example:**
```
ALTER TABLE students ADD COLUMN favorite_subject VARCHAR(255);
```
Comment: Adds a new column named 'favorite_subject' to the 'students' table.

7. WHERE

* **Purpose:** Filters the results of a query based on a condition.
* **Example:**
```
SELECT * FROM students WHERE age > 20;
```
Comment: Selects all students with age greater than 20 from the 'students' table.

8. FROM

* **Purpose:** Specifies the table(s) from which data will be retrieved or modified.
* **Example:**
```
SELECT * FROM students;
```
Comment: Selects all rows from the 'students' table.

9. GROUP BY

* **Purpose:** Groups the results of a query by one or more columns.
* **Example:**
```
SELECT COUNT(*) AS student_count, department FROM students
GROUP BY department;
```
Comment: Counts the number of students in each department in the 'students' table.

10. HAVING

* **Purpose:** Filters the results of a GROUP BY query based on a condition.
* **Example:**
```
SELECT department, COUNT(*) AS student_count FROM students
GROUP BY department HAVING COUNT(*) > 5;
```
Comment: Selects departments with more than 5 students from the 'students' table.

11. ORDER BY

* **Purpose:** Sorts the results of a query by one or more columns in ascending or descending order.
* **Example:**
```
SELECT * FROM students ORDER BY name ASC;
```
Comment: Selects all students from the 'students' table and sorts them by name in ascending order.

12. ASC

* **Purpose:** Specifies that the results of a query should be sorted in ascending order (smallest to largest).
* **Example:**
```
SELECT * FROM students ORDER BY name ASC;
```
Comment: Sorts students by name in ascending order.

13. LIMIT

* **Purpose:** Limits the number of rows returned by a query.
* **Example:**
```
SELECT * FROM students LIMIT 10;
```
Comment: Selects the first 10 rows from the 'students' table.

14. IN

* **Purpose:** Checks if a value is present in a specified list of values.
* **Example:**
```
SELECT * FROM students WHERE name IN ('John Doe', 'Jane Smith');
```

```
**Comment:** Selects students whose names are either 'John Doe' or 'Jane Smith' from the 'students' table.

**15. BETWEEN ... AND ...**

* **Purpose:** Checks if a value falls within a specified range of values.
* **Example:**
```
SELECT * FROM students WHERE age BETWEEN 20 AND 25;
```
**Comment:** Selects students whose ages are between 20 and 25 (inclusive) from the 'students' table.

**16. LIKE**

* **Purpose:** Performs a pattern matching operation on a string value.
* **Example:**
```
SELECT * FROM students WHERE name LIKE '%Doe';
```
**Comment:** Selects students whose names contain the substring 'Doe' from the 'students' table.

**17. MAX()**

* **Purpose:** Returns the maximum value of a column in a group of rows.
* **Example:**
```
SELECT MAX(age) FROM students;
```
**Comment:** Retrieves the maximum age of all students in the 'students' table.

**PostgreSQL Functions and Keywords for Writing Queries (Continued)**
```

18. MIN()

* **Purpose:** Returns the minimum value of a column in a group of rows.
* **Example:**
```
SELECT MIN(age) FROM students;
```
Comment: Retrieves the minimum age of all students in the 'students' table.

19. AVG()

* **Purpose:** Returns the average value of a column in a group of rows.
* **Example:**
```
SELECT AVG(age) FROM students;
```
Comment: Retrieves the average age of all students in the 'students' table.

20. SUM()

* **Purpose:** Returns the sum of values in a column in a group of rows.
* **Example:**
```
SELECT SUM(age) FROM students;
```
Comment: Retrieves the total age of all students in the 'students' table.

21. COUNT

* **Purpose:** Returns the number of rows in a table or the number of times a specific value appears.
* **Example:**

```
SELECT COUNT(*) FROM students;
```
Comment: Retrieves the total number of students in the 'students' table.

22. ADD CONSTRAINT

* **Purpose:** Adds a constraint to a table, such as a primary key or foreign key.
* **Example:**
```
ALTER TABLE students ADD CONSTRAINT pk_student PRIMARY KEY
(student_id);
```
Comment: Adds a primary key constraint on the 'student_id' column in the 'students' table.

23. DROP

* **Purpose:** Removes a table, column, or constraint from the database.
* **Example:**
```
DROP TABLE students;
```
Comment: Drops the 'students' table from the database.

24. FOREIGN KEY

* **Purpose:** Specifies a relationship between two tables, ensuring that data integrity is maintained.
* **Example:**
```
ALTER TABLE orders ADD FOREIGN KEY (student_id) REFERENCES
students(student_id);
```

Comment: Creates a foreign key constraint on the 'student_id' column in the 'orders' table, referencing the 'student_id' primary key in the 'students' table.

25. PRIMARY KEY

* **Purpose:** Specifies a unique column or set of columns that uniquely identifies each row in a table.
* **Example:**
```
ALTER TABLE students ADD PRIMARY KEY (student_id);
```
Comment: Adds a primary key constraint on the 'student_id' column in the 'students' table.

26. REFERENCES

* **Purpose:** Specifies the table and column that a foreign key references.
* **Example:**
```
ALTER TABLE orders ADD FOREIGN KEY (student_id) REFERENCES students(student_id);
```
Comment: Specifies that the 'student_id' foreign key in the 'orders' table references the 'student_id' primary key in the 'students' table.

27. ON DELETE

* **Purpose:** Specifies the action to be taken when a row is deleted from the referenced table.
* **Example:**
```
ALTER TABLE orders ADD FOREIGN KEY (student_id) REFERENCES students(student_id) ON DELETE CASCADE;
```

Comment: Specifies that when a student is deleted from the 'students' table, all orders associated with that student should also be deleted from the 'orders' table.

28. ON UPDATE

* **Purpose:** Specifies the action to be taken when a row is updated in the referenced table.
* **Example:**
```
ALTER TABLE orders ADD FOREIGN KEY (student_id) REFERENCES students(student_id) ON UPDATE CASCADE;
```
Comment: Specifies that when a student's ID changes in the 'students' table, the 'student_id' foreign key in the 'orders' table should also be updated.

29. ON

* **Purpose:** Used in conjunction with ON DELETE or ON UPDATE to specify the referenced table and column.
* **Example:**
```
ALTER TABLE orders ADD FOREIGN KEY (student_id) REFERENCES students(student_id) ON DELETE CASCADE;
```
Comment: Specifies that the 'student_id' foreign key in the 'orders' table references the 'student_id' primary key in the 'students' table, and that when a student is deleted from the 'students' table, all orders associated with that student should also be deleted from the 'orders' table.

30. CASCADE

* **Purpose:** Specifies that the action specified in ON DELETE or ON UPDATE should be performed automatically.

* **Example:**
```
ALTER TABLE orders ADD FOREIGN KEY (student_id) REFERENCES students(student_id) ON DELETE CASCADE;
```
Comment: Specifies that when a student is deleted from the 'students' table, all orders associated with that student should also be deleted from the 'orders' table, without requiring any additional action.

31. UNIQUE

* **Purpose:** Specifies that a column or set of columns must contain unique values.
* **Example:**
```
ALTER TABLE students ADD UNIQUE (name);
```
Comment: Adds a unique constraint on the 'name' column in the 'students' table, ensuring that no two students can have the same name.

32. CREATE INDEX

* **Purpose:** Creates an index on a column or set of columns to improve query performance.
* **Example:**
```
CREATE INDEX idx_student_name ON students(name);
```
Comment: Creates an index on the 'name' column in the 'students' table, which can be used to speed up queries that filter or sort by student name.

33. JOIN

* **Purpose:** Combines rows from two or more tables based on a common column or columns.
* **Example:**
```
SELECT * FROM students JOIN orders ON students.student_id =
orders.student_id;
```
Comment: Joins the 'students' and 'orders' tables on the 'student_id' column, returning all rows from both tables that have matching student IDs.

34. INNER

* **Purpose:** Specifies that only rows that have matching values in both tables should be included in the join result.
* **Example:**
```
SELECT * FROM students INNER JOIN orders ON students.student_id =
orders.student_id;
```
Comment: Performs an inner join between the 'students' and 'orders' tables, returning only the rows where the student ID matches in both tables.

PostgreSQL Functions and Keywords for Writing Queries (Continued)

35. LEFT

Purpose: Specifies that all rows from the left table should be included in the join result, even if there are no matching rows in the right table.
Example:
SELECT * FROM students LEFT JOIN orders ON students.student_id =
orders.student_id;

Comment: Performs a left join between the 'students' and 'orders' tables, returning all students, even if they have no associated orders.

36. RIGHT

Purpose: Specifies that all rows from the right table should be included in the join result, even if there are no matching rows in the left table.
Example:
SELECT * FROM students RIGHT JOIN orders ON students.student_id = orders.student_id;

Comment: Performs a right join between the 'students' and 'orders' tables, returning all orders, even if they are not associated with any students.

37. UNION

Purpose: Combines the results of two or more SELECT statements into a single result set, removing duplicate rows.
Example:
SELECT * FROM students UNION SELECT * FROM teachers;

Comment: Combines the results of two SELECT statements that retrieve data from the 'students' and 'teachers' tables, removing any duplicate rows.

38. UNION ALL

Purpose: Combines the results of two or more SELECT statements into a single result set, including duplicate rows.
Example:
SELECT * FROM students UNION ALL SELECT * FROM teachers;

Comment: Combines the results of two SELECT statements that retrieve data from the 'students' and 'teachers' tables, including any duplicate rows.

39. QUOTE()

Purpose: Encloses a string in single quotes, which is useful for including special characters or reserved words in a query.

Example:

```
SELECT * FROM students WHERE name = QUOTE('O''Brien');
```

Comment: Retrieves students whose names contain an apostrophe, such as 'O'Brien'.

40. LENGTH()

Purpose: Returns the length of a string.
Example:

```
SELECT LENGTH('Hello World');
```

Comment: Returns the length of the string 'Hello World', which is 11 characters.

41. POSITION()

Purpose: Returns the starting position of a substring within a string.
Example:

```
SELECT POSITION('World' IN 'Hello World');
```

Comment: Returns the position of the substring 'World' in the string 'Hello World', which is 6.

42. STRCMP()

Purpose: Compares two strings and returns an integer indicating their relative order.
Example:

```
SELECT STRCMP('Hello', 'World');
```

Comment: Compares the strings 'Hello' and 'World' and returns -1, indicating that 'Hello' comes before 'World' in alphabetical order.

43. CONCAT()

Purpose: Concatenates two or more strings into a single string.
Example:
```
SELECT CONCAT('Hello', ' ', 'World');
```

Comment: Concatenates the strings 'Hello', ' ', and 'World' into the single string 'Hello World'.

44. REPLACE()

Purpose: Replaces all occurrences of a substring with another substring.
Example:
```
SELECT REPLACE('Hello World', 'World', 'Universe');
```

Comment: Replaces all occurrences of the substring 'World' with the substring 'Universe' in the string 'Hello World'.

45. SUBSTRING()

Purpose: Extracts a substring from a string, starting at a specified position and continuing for a specified length.
Example:
```
SELECT SUBSTRING('Hello World', 6, 5);
```

Comment: Extracts the substring 'World' from the string 'Hello World', starting at position 6 and continuing for 5 characters.

46. ACOS()

Purpose: Returns the arccosine of a number.
Example:
```
SELECT ACOS(0.5);
```

Comment: Returns the arccosine of 0.5, which is approximately 1.047 radians.

47. ASIN()

Purpose: Returns the arcsine of a number.
Example:
```
SELECT ASIN(0.5);
```

Comment: Returns the arcsine of 0.5, which is approximately 0.523 radians.

48. COS()

Purpose: Returns the cosine of an angle specified in radians.
Example:

```
SELECT COS(PI/2);
```

Comment: Returns the cosine of pi/2 radians, which is 0.

49. SIN()

Purpose: Returns the sine of an angle specified in radians.
Example:
```
SELECT SIN(PI/2);
```

Comment: Returns the sine of pi/2 radians, which is 1.

50. COT()

Purpose: Returns the cotangent of an angle specified in radians.
Example:
```
SELECT COT(PI/4);
```

Comment: Returns the cotangent of pi/4 radians, which is 1.

51. TAN()

 Purpose: Returns the tangent of an angle specified in radians.
 Example:
SELECT TAN(PI/4);

Comment: Returns the tangent of pi/4 radians, which is 1.

PostgreSQL Functions and Keywords for Writing Queries (Continued)

52. MOD()

* **Purpose:** Returns the remainder of dividing one number by another.
* **Example:**
```
SELECT MOD(10, 3);
```
Comment: Returns the remainder of dividing 10 by 3, which is 1.

53. POW()

* **Purpose:** Raises a number to a specified power.
* **Example:**
```
SELECT POW(2, 3);
```
Comment: Raises 2 to the power of 3, which is 8.

54. CEIL()

* **Purpose:** Rounds a number up to the nearest integer.
* **Example:**
```
SELECT CEIL(3.14);
```

Comment: Rounds 3.14 up to the nearest integer, which is 4.

55. FLOOR()

* **Purpose:** Rounds a number down to the nearest integer.
* **Example:**
```
SELECT FLOOR(3.14);
```
Comment: Rounds 3.14 down to the nearest integer, which is 3.

56. ROUND()

* **Purpose:** Rounds a number to the nearest specified number of decimal places.
* **Example:**
```
SELECT ROUND(3.14159, 2);
```
Comment: Rounds 3.14159 to 2 decimal places, which is 3.14.

57. TRUNCATE()

* **Purpose:** Truncates a number to a specified number of decimal places.
* **Example:**
```
SELECT TRUNCATE(3.14159, 2);
```
Comment: Truncates 3.14159 to 2 decimal places, which is 3.14.

58. SIGN()

* **Purpose:** Returns the sign of a number (-1, 0, or 1) indicating whether it is negative, zero, or positive.

* **Example:**
```
SELECT SIGN(-1);
```
Comment: Returns -1, indicating that the number is negative.

59. ABS()

* **Purpose:** Returns the absolute value of a number.
* **Example:**
```
SELECT ABS(-1);
```
Comment: Returns 1, which is the absolute value of -1.

60. STRING_AGG()

* **Purpose:** Concatenates the values of a column into a single string, separated by a specified delimiter.
* **Example:**
```
SELECT STRING_AGG(name, ', ') FROM students;
```
Comment: Concatenates the names of all students into a single string, separated by commas.

61. DISTINCT

* **Purpose:** Removes duplicate rows from a query result.
* **Example:**
```
SELECT DISTINCT name FROM students;
```
Comment: Returns a list of unique student names, removing any duplicates.

62. CASE

* **Purpose:** Evaluates a series of conditions and returns a different value for each condition that is met.
* **Example:**
```
SELECT CASE
  WHEN age < 18 THEN 'Minor'
  WHEN age >= 18 AND age < 65 THEN 'Adult'
  ELSE 'Senior'
END
FROM students;
```
Comment: Classifies students as 'Minor', 'Adult', or 'Senior' based on their age.

63. WHEN

* **Purpose:** Specifies a condition to be evaluated in a CASE statement.
* **Example:**
```
SELECT CASE
  WHEN age < 18 THEN 'Minor'
  WHEN age >= 18 AND age < 65 THEN 'Adult'
  ELSE 'Senior'
END
FROM students;
```
Comment: Specifies the condition that determines whether a student is classified as 'Minor', 'Adult', or 'Senior'.

64. ELSE

* **Purpose:** Specifies the value to be returned if none of the conditions in a CASE statement are met.
* **Example:**
```
SELECT CASE
  WHEN age < 18 THEN 'Minor'
  WHEN age >= 18 AND age < 65 THEN 'Adult'
  ELSE 'Senior'
END
FROM students;
```
Comment: Specifies that if a student's age does not meet any of the other conditions, they should be classified as 'Senior'.

65. END

* **Purpose:** Ends a CASE statement.
* **Example:**
```
SELECT CASE
  WHEN age < 18 THEN 'Minor'
  WHEN age >= 18 AND age < 65 THEN 'Adult'
  ELSE 'Senior'
END
FROM students;
```
Comment: Ends the CASE statement and returns the appropriate classification for each student.

66. IF

* **Purpose:** Evaluates a condition and executes a statement if the condition is met.
* **Example:**
```
```

IF age >= 18 THEN
  UPDATE students SET is_adult = true;
END IF;
```
**Comment:** Updates the 'is_adult' column to true for all students who are 18 years of age or older.

**67. THEN**

* **Purpose:** Specifies the statement to be executed if the condition in an IF statement is met.
* **Example:**
```
IF age >= 18 THEN
  UPDATE students SET is_adult = true;
END IF;
```
**Comment:** Specifies that the 'is_adult' column should be updated to true if the student's age is 18 or older.

**68. BEGIN**

* **Purpose:** Starts a block of code that can contain multiple statements.
* **Example:**
```
BEGIN
  UPDATE students SET is_adult = true WHERE age >= 18;
  INSERT INTO adults (student_id) SELECT student_id FROM students
WHERE age >= 18;
END;
```
**Comment:** Updates the 'is_adult' column for students who are 18 or older and inserts their IDs into the 'adults' table.

PostgreSQL Functions and Keywords for Writing Queries (Continued)
```

69. COMMIT

Purpose: Makes permanent all changes made to the database since the last COMMIT or ROLLBACK.

Example:

```
BEGIN;
UPDATE students SET age = age + 1;
COMMIT;
```

Comment: Updates the age of all students and makes the change permanent.

70. ROLLBACK

Purpose: Reverts all changes made to the database since the last COMMIT or ROLLBACK.

Example:

```
BEGIN;
UPDATE students SET age = age + 1;
ROLLBACK;
```

Comment: Updates the age of all students, but then reverts the change, leaving the ages unchanged.

71. SAVEPOINT

Purpose: Creates a named point in time to which you can roll back if needed.

Example:

```
BEGIN;
SAVEPOINT my_savepoint;
UPDATE students SET age = age + 1;
-- If something goes wrong, you can rollback to the savepoint
IF something_went_wrong THEN
```

```
  ROLLBACK TO my_savepoint;
END IF;
COMMIT;
```

Comment: Creates a savepoint named 'mysavepoint' before updating the ages of students. If something goes wrong with the update, you can roll back to the savepoint to revert the changes.

72. ROLLBACK TO

* **Purpose:** Rolls back all changes made to the database since the specified savepoint.
* **Example:**
```
BEGIN;
SAVEPOINT mysavepoint;
UPDATE students SET age = age + 1;
-- If something goes wrong, you can rollback to the savepoint
IF somethingwentwrong THEN
  ROLLBACK TO mysavepoint;
END IF;
COMMIT;
```
Comment: Rolls back all changes made since the savepoint named 'mysavepoint' was created.

73. CREATE (OR REPLACE) PROCEDURE

Purpose: Creates or replaces a stored procedure, which is a set of SQL statements that can be executed as a unit.
Example:
```
CREATE PROCEDURE update_student_age(IN student_id INT, IN
new_age INT) AS
BEGIN
  UPDATE students SET age = new_age WHERE student_id = student_id;
```

END;

Comment: Creates a stored procedure named 'updatestudentage' that takes two input parameters: the student ID and the new age. The procedure updates the age of the student with the specified ID.

74. DEFAULT

Purpose: Specifies the default value for a column when no value is provided.
Example:
```
CREATE TABLE students (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  age INT DEFAULT 18
);
```

Comment: Creates a table named 'students' with a column named 'age' that has a default value of 18. If no age is specified when inserting a new student, the age will be set to 18 by default.

75. VALUES

Purpose: Used in INSERT statements to specify the values to be inserted into a table.
Example:
```
INSERT INTO students (name, age) VALUES ('John Doe', 21);
```

Comment: Inserts a new student named 'John Doe' with age 21 into the 'students' table.

76. CREATE OR REPLACE FUNCTION

Purpose: Creates or replaces a stored function, which is a set of SQL statements that returns a value.

Example:

```
CREATE OR REPLACE FUNCTION get_student_name(IN student_id INT)
RETURNS VARCHAR(255) AS
BEGIN
  SELECT name FROM students WHERE student_id = student_id;
END;
```

Comment: Creates or replaces a stored function named 'getstudentname' that takes a student ID as input and returns the student's name.

77. RETURN

Purpose: Used in stored functions to return a value to the caller.
Example:

```
CREATE OR REPLACE FUNCTION get_student_name(IN student_id INT)
RETURNS VARCHAR(255) AS
BEGIN
  SELECT name FROM students WHERE student_id = student_id;
END;
```

Comment: Returns the student's name from the stored function 'getstudentname'.

78. CREATE TRIGGER

Purpose: Creates a trigger, which is a set of SQL statements that are executed automatically when a specific event occurs on a table.
Example:

```
CREATE TRIGGER update_student_age_on_insert
AFTER INSERT ON students
FOR EACH ROW
BEGIN
  UPDATE students SET age = age + 1 WHERE student_id =
NEW.student_id;
END;
```

Comment: Creates a trigger named 'updatestudentageoninsert' that is fired after a new row is inserted into the 'students' table. The trigger increments the age of the newly inserted student by 1.

79. FOR EACH ROW

Purpose: Specifies that the trigger should be executed for each row that is affected by the event.
Example:
```
CREATE TRIGGER update_student_age_on_insert
AFTER INSERT ON students
FOR EACH ROW
BEGIN
  UPDATE students SET age = age + 1 WHERE student_id =
NEW.student_id;
END;
```

Comment: Specifies that the 'updatestudentageoninsert' trigger should be executed for each new row that is inserted into the 'students' table.

80. EXECUTE PROCEDURE

Purpose: Executes a stored procedure.
Example:
```
EXECUTE PROCEDURE update_student_age(123, 22);
```

Comment: Executes the 'updatestudentage' stored procedure with the student ID 123 and the new age 22.

81. ROWNUMBER()**

* **Purpose:** Returns the sequential number of a row within a partition of a result set.
* **Example:**

```
SELECT ROWNUMBER() OVER (PARTITION BY department ORDER BY name) AS rownum, name
FROM students;
```
Comment: Assigns a sequential number to each student within each department, ordered by name.

82. RANK()

* **Purpose:** Returns the rank of a row within a partition of a result set, ignoring duplicate values.
* **Example:**
```
SELECT RANK() OVER (PARTITION BY department ORDER BY name) AS rank, name
FROM students;
```
Comment: Assigns a rank to each student within each department, ignoring duplicate names.

**83. FIRSTVALUE()

Purpose: Returns the first value of a column within a partition of a result set, ignoring null values.
Example:
```
SELECT FIRST_VALUE(name) OVER (PARTITION BY department ORDER BY name) AS first_name
FROM students;
```

Comment: Retrieves the first non-null name for each department.

84. LAG()

Purpose: Returns the value of a column from the previous row in a result set.
Example:
```
SELECT name, LAG(name) OVER (ORDER BY name) AS previous_name
FROM students;
```

Comment: Retrieves the name of each student along with the name of the previous student in alphabetical order.

85. LEAD()

Purpose: Returns the value of a column from the next row in a result set.
Example:
```
SELECT name, LEAD(name) OVER (ORDER BY name) AS next_name
FROM students;
```

Comment: Retrieves the name of each student along with the name of the next student in alphabetical order.

PostgreSQL Functions and Keywords for Writing Queries (Continued)

86. OVER()

* **Purpose:** Specifies a window function, which operates on a set of rows that are related to the current row.
* **Example:**
```
SELECT name, RANK() OVER (PARTITION BY department ORDER BY
name) AS rank
FROM students;
```
Comment: Calculates the rank of each student within their department, using the OVER() clause to specify the window function.

87. PARTITION BY

* **Purpose:** Divides the result set into groups, or partitions, based on the specified column(s).
* **Example:**
```
SELECT name, RANK() OVER (PARTITION BY department ORDER BY name) AS rank
FROM students;
```
Comment: Partitions the students by department before calculating their ranks.

88. GROUP_CONCAT()

* **Purpose:** Concatenates the values of a column into a single string, separated by a specified delimiter, for each group in a result set.
* **Example:**
```
SELECT department, GROUP_CONCAT(name) AS student_names
FROM students
GROUP BY department;
```
Comment: Concatenates the names of students within each department into a single string, separated by commas.