Interview Prep

`MAX()` Aggregate Function

The `MAX()` aggregate function takes the name of a column as an argument and returns the largest value in a column. The given query will return the largest value from the `amount` column.

```
SELECT MAX(amount)
FROM transactions;
```

`SELECT` Statement

The `SELECT *` statement returns all columns from the provided table in the result set. The given query will fetch all columns and records (rows) from the `movies` table.

```
SELECT *
FROM movies;
```

`ORDER BY` Clause

The `ORDER BY` clause can be used to sort the result set by a particular column either alphabetically or numerically. It can be ordered in two ways:

- `DESC` is a keyword used to sort the results in descending order.
- `ASC` is a keyword used to sort the results in ascending order (default).

```
SELECT *
FROM contacts
ORDER BY birth_date DESC;
```

`COUNT()` Aggregate Function

The `COUNT()` aggregate function returns the total number of rows that match the specified criteria. For instance, to find the total number of employees who have less than 5 years of experience, the given query can be used.

Note: A column name of the table can also be used instead of `*`. Unlike `COUNT(*)`, this variation `COUNT(column)` will not count `NULL` values in that column.

```
SELECT COUNT(*)
FROM employees
WHERE experience < 5;
```

`DISTINCT` Clause

Unique values of a column can be selected using a `DISTINCT` query. For a table `contact_details` having five rows in which the `city` column contains Chicago, Madison, Boston, Madison, and Denver, the given query would return:

- Chicago
- Madison

```
SELECT DISTINCT city
FROM contact_details;
```

- Boston
- Denver

`LIMIT` Clause

The `LIMIT` clause is used to narrow, or limit, a result set to the specified number of rows. The given query will limit the result set to 5 rows.

```
SELECT *
FROM movies
LIMIT 5;
```

`GROUP BY` Clause

The `GROUP BY` clause will group records in a result set by identical values in one or more columns. It is often used in combination with aggregate functions to query information of similar records. The `GROUP BY` clause can come after `FROM` or `WHERE` but must come before any `ORDER BY` or `LIMIT` clause.
The given query will count the number of movies per rating.

```
SELECT rating,
   COUNT(*)
FROM movies
GROUP BY rating;
```

`MIN()` Aggregate Function

The `MIN()` aggregate function returns the smallest value in a column. For instance, to find the smallest value of the `amount` column from the table named `transactions`, the given query can be used.

```
SELECT MIN(amount)
FROM transactions;
```

CASE statement in SQL

The SQL `CASE` statement enables control flow in SQL. It allows for one or more conditions (`WHEN` condition `THEN` result) and an optional default case (`ELSE`). The query above will provide each rating a value for the specified ranges within the result set.

```
SELECT name,
  CASE
    WHEN rating > 8 THEN "Excellent"
    WHEN rating > 5 THEN "Good"
    WHEN rating > 3 THEN "Okay"
    ELSE "Bad"
  END
FROM movies;
```

`HAVING` Clause

The `HAVING` clause is used to further filter the result set groups provided by the `GROUP BY` clause. `HAVING` is often used with aggregate functions to filter the result set groups based on an aggregate property. The given query will select only the records (rows) from only years where more than 5 movies were released per year.

```
SELECT year,
   COUNT(*)
FROM movies
GROUP BY year
HAVING COUNT(*) > 5;
```

The `HAVING` clause must always come after a `GROUP BY` clause but must come before any `ORDER BY` or `LIMIT` clause.

WHERE Clause

The `WHERE` clause is used to filter records (rows) that match a certain condition. The given query will select all records where the `pub_year` equals `2017`.

```
SELECT title
FROM library
WHERE pub_year = 2017;
```

ROUND() Function

The `ROUND()` function will round a number value to a specified number of places. It takes two arguments: a number, and a number of decimal places. It can be combined with other aggregate functions, as shown in the given query. This query will calculate the average rating of movies from 2015, rounding to 2 decimal places.

```
SELECT year,
   ROUND(AVG(rating), 2)
FROM movies
WHERE year = 2015;
```

Outer Join

An outer join will combine rows from different tables even if the join condition is not met. In a `LEFT JOIN`, every row in the *left* table is returned in the result set, and if the join condition is not met, then `NULL` values are used to fill in the columns from the *right* table.

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
  ON table1.column_name =
table2.column_name;
```

Inner Join

The `JOIN` clause allows for the return of results from more than one table by joining them together with other results based on common column values specified using an `ON` clause. `INNER JOIN` is the default `JOIN` and it will only return results matching the condition specified by `ON`.

```
SELECT *
FROM books
JOIN authors
  ON books.author_id = authors.id;
```

Column Constraints

Column constraints are the rules applied to the values of individual columns:

- `PRIMARY KEY` constraint can be used to uniquely identify the row.
- `UNIQUE` columns have a different value for every row.
- `NOT NULL` columns must have a value.
- `DEFAULT` assigns a default value for the column when no value is specified.

```
CREATE TABLE student (
  id INTEGER PRIMARY KEY,
  name TEXT UNIQUE,
  grade INTEGER NOT NULL,
  age INTEGER DEFAULT 10
);
```

There can be only one `PRIMARY KEY` column per table and multiple `UNIQUE` columns.

ALTER TABLE Statement

The `ALTER TABLE` statement is used to modify the columns of an existing table. When combined with the `ADD COLUMN` clause, it is used to add a new column.

```
ALTER TABLE table_name
ADD column_name datatype;
```

Primary Key

A *primary key* column in a SQL table is used to uniquely identify each record in that table. A primary key cannot be `NULL`. In the example, `customer_id` is the primary key. The same value cannot re-occur in a primary key column. Primary keys are often used in `JOIN` operations.

customer_id	f_name	l_name
1	Abby	Caren
2	Aaron	Paul
3	Gratian	Joseph

UPDATE Statement

The `UPDATE` statement is used to edit records (rows) in a table. It includes a `SET` clause that indicates the column to edit and a `WHERE` clause for specifying the record(s).

```
UPDATE table_name
SET column1 = value1, column2 = value2
WHERE some_column = some_value;
```

Data Types As Constraints

Columns of a PostgreSQL database table must have a data type, which constrains the type of information that can be entered into that column. This is important in order to ensure data integrity and consistency over time. Some common PostgreSQL types are `integer`, `decimal`, `varchar`, and `boolean`. Data types are defined in a `CREATE TABLE` statement by indicating the data type after each column name.

```
CREATE TABLE tablename (
    myNum integer,
    myString varchar(50)
);
```

NOT NULL

In PostgreSQL, `NOT NULL` constraints can be used to ensure that particular columns of a database table do not contain missing data. This is important for ensuring database integrity and consistency over time. `NOT NULL` constraints can be enforced within a `CREATE TABLE` statement using `NOT NULL`.

```
CREATE TABLE table_name (
    column_1 integer NOT NULL,
    column_2 text NOT NULL,
    column_3 numeric
);
```

UNIQUE

In PostgreSQL, `UNIQUE` constraints can be used to ensure that elements of a particular column (or group of columns) are unique (i.e., no two rows have the same value or combination of values). This is important for ensuring database integrity and consistency over time. `UNIQUE` constraints can be enforced within a `CREATE TABLE` statement using the `UNIQUE` keyword.

```
CREATE TABLE table_name (
    column_1 integer UNIQUE,
    column_2 text UNIQUE,
    column_3 numeric,
    column_4 text,
    UNIQUE(column_3, column_4)
);
```

Primary Keys

The primary key of a database table is a column or group of columns that can be used to uniquely identify every row of the table. For example, a table of students might have a primary key named `student_id`, which contains unique ID numbers for each student.

One-to-One Database Relationships

In a relational database, two tables have a one-to-one relationship if each row in one table links to exactly one row in the other table, and vice versa. For example, a table of `employees` and a table of `employee_contact_info` might have a one-to-one relationship if every employee listed in the `employees` table has contact information listed in the `employee_contact_info` table and vice versa.