SQL Cheat Sheet

Background: What is SQL? Why do we need it?

SQL is a database language used to query and manipulate the data in the database.

Main objectives:

- To provide an efficient and convenient environment
- Manage information about users who interact with the DBMS

The SQL statements can be categorized as

Data Definition Language(DDL) Commands:

- CREATE: creates a new database object, such as a table.
- ALTER: used to modify the database object
- DROP: used to delete the objects.

Data Manipulation Language(DML) Commands:

- INSERT: used to insert a new data row record in a table.
- UPDATE: used to modify an existing record in a table.
- DELETE: used delete a record from the table.

Data Control Language(DCL) Commands:

- GRANT: used to assign permission to users to access database objects.
- REVOKE: used to deny permission to users to access database objects.

Data Query Language(DQL) Commands:

- SELECT: it is the DQL command to select data from the database.

Data Transfer Language(DTL) Commands:

- COMMIT: used to save any transaction into the database permanently.
- ROLLBACK: restores the database to the last committed state.

Identifying Data Types

Data types specify the type of data that an object can contain, such as integer data or character data. We need to specify the data type according to the data to be stored.

Following are some of the essential data types:

Data Type	Used to Store
int	Integer data
smallint	Integer data

tinyint	Integer data
bigint	Integer data
decimal	Numeric data type with a fixed precision and scale.
numeric	numeric data type with a fixed precision and scale.
float	floating precision data
money	monetary data
datetime	data and time data
char(n)	fixed length character data
varchar(n)	variable length character data
text	character string
bit	integer data with 0 or 1
image	variable length binary data to store images
real	floating precision number
binary	fixed length binary data
cursor	cursor reference
sql_variant	different data types
timestamp	unique number in the database that is updated every time in a row that contains timestamp is inserted or updated.
table	temporary set of rows returned as a result set of a table-valued function.
xml	store and return xml values

Managing Tables

Create Table

Table can be created using the CREATE TABLE statement. The syntax is as follows:

```
CREATE TABLE table_name

( col_name1 datatype,

col_name2 datatype,

col_name3 datatype,

…

);
```

Example: Create a table named EmployeeLeave in Human Resource schema with the following attributes:

Columns	Data Type	Checks
EmployeeID	int	NOT NULL
LeaveStartDate	date	NOT NULL
LeaveEndDate	date	NOT NULL
LeaveReason	varchar(100)	NOT NULL
LeaveType	char(2)	NOT NULL

```
CREATE TABLE HumanResources.EmployeeLeave

(

EmployeeID int NOT NULL,

LeaveStartDate datetime NOT NULL,

LeaveEndDate datetime NOT NULL,

LeaveReason varchar(100),

LeaveType char(2) NOT NULL );
```

Constraints in SQL

Constraints define rules that must be followed to maintain consistency and correctness of data. A constraint can be created by using either of the following statements:

```
CREATE TABLE statement

ALTER TABLE statement
```

```
CREATE TABLE table_name

(

column_name CONSTRAINT constraint_name constraint_type

)
```

Types of Constraints:

Constraint	Description	Syntax
Primary key	Columns or columns that uniquely identify all rows in the table.	CREATE TABLE table_name (col_name [CONSTRAINT constraint_name PRIMARY KEY] (col_name(s)))

Unique key	Enforces uniqueness on non primary key columns.	`CREATE TABLE table_name`
		`( col_name [CONSTRAINT constraint_name UNIQUE KEY] (col_name(s))`
		`)`

Foreign key	Is used to remove the inconsistency in two tables when the data depends on other tables.	`CREATE TABLE table_name`
		`( col_name [CONSTRAINT constraint_name FOREIGN KEY] (col_name)`
		`REFERENCES table_name (col_name)`
		`)`

Check	Enforce domain integrity by restricting the values to be inserted in the column.	`CREATE TABLE table_name`
		`( col_name [CONSTRAINT constraint_name] CHECK (expression) (col_name(s)) )`
		`expression:`
		`IN, LIKE, BETWEEN`

3.2 Modifying Tables

Modify table using ALTER TABLE statement when:

1. Adding column
2. Altering data type
3. Adding or removing constraints

Syntax of ALTER TABLE:

```
ALTER TABLE table_name

ADD column_name;

ALTER TABLE table_name

DROP COLUMN column_name;

ALTER TABLE table_name

ALTER COLUMN column_name data_type;
```

Renaming a Table

A table can be renamed whenever required using RENAME TABLE statement:

RENAME TABLE old_table_name TO new_table_name;

Dropping a Table versus Truncate Table

A table can be dropped or deleted when no longer required using DROP TABLE statement:

```
DROP TABLE table_name;
```

The contents of the table can be deleted when no longer required without deleting the table itself using TRUNCATE TABLE statement:

```
TRUNCATE TABLE table_name;
```

Manipulating Data

Storing Data in a Table

Syntax:

```
INSERT INTO table_name (col_name1, col_name2, col_name3…)

VALUES (value1, value2, value3…);
```

Example: Inserting data into Student table.

```
INSERT INTO Student (StudentID, FirstName,LastName,Marks)
```

```
VALUES (‘101’,’John’,’Ray’,’78’);
```

Example: Inserting multiple data into Student table.

```
INSERT INTO Student

VALUES (101,’John’,’Ray’,78),

 (102,‘Steve’,’Jobs’,89),

 (103,‘Ben’,’Matt’,77),

 (104,‘Ron’,’Neil’,65),

 (105,‘Andy’,’Clifton’,65),

 (106,‘Park’,’Jin’,90);
```

Copying Data from one table to another:

```
INSERT INTO table_name2

SELECT * FROM table_name1

WHERE [condition]
```

Updating Data in a Table

Data can be updated in the table using UPDATE DML statement:

```
SELECT table_name

SET col_name1 = value1 , col_name2 = value2…

WHERE condition
```

Example update marks of Andy to 85

```
SELECT table_name

SET Marks = 85

WHERE FirstName = ‘Andy’
```

Deleting Data from a Table

A row can be deleted when no longer required using DELETE DML statement.

Syntax:

```
DELETE FROM table_name

WHERE condition

DELETE FROM Student

WHERE StudentID = ‘103’
```

Deleting all records from a table:

```
DELETE table_name
```

Retrieving Attributes

One or more column can be displayed while retrieving data from the table.

One may want to view all the details of the Employee table or might want to view few columns.

Required data can be retrieved data from the database tables by using the SELECT statement.

The syntax of SELECT statement is:

```
SELECT [ALL | DISTINCT] select_column_list

[INTO [new_table_name]]

[FROM [table_name | view_name]]

[WHERE search condition]
```

Consider the following Student table:

StudentID	FirstName	LastName	Marks
101	John	Ray	78
102	Steve	Jobs	89
103	Ben	Matt	77
104	Ron	Neil	65
105	Andy	Clifton	65
106	Park	Jin	90

Retrieving Selected Rows

To retrieve selected rows from a table use WHERE clause in the SELECT statement.

```
SELECT *

FROM Student

WHERE StudentID = 104;
```

HAVING Clause is used instead of WHERE for aggregate functions.

Comparison Operators

Comparison operators test for the similarity between two expressions.

Syntax:

```
SELECT column_list

FROM table_name

WHERE expression1 comparison_operatore expression2
```

Example of some comparison operators:

```
SELECT StudentID,Marks

FROM Student

WHERE Marks = 90;

SELECT StudentID,Marks

FROM Student

WHERE StudentID > 101;

SELECT StudentID,Marks

FROM Student

WHERE Marks != 89;

SELECT StudentID,Marks

FROM Student

WHERE Marks >= 50;
```

Logical Operators

Logical operators are used to SELECT statement to retrieve records based on one or more conditions. More than one logical operator can be combined to apply multiple search conditions.

Syntax:

```
SELECT column_list

FROM table_name

WHERE conditional_expression1 [NOT]

 conditional_expression2
```

Types of Logical Operators:

OR Operator

```
SELECT StudentID,Marks,

FROM Student

WHERE Marks= 40 OR Marks=56 OR Marks = 65;
```

AND Operator

```
SELECT StudentID,Marks,

FROM Student

WHERE Marks= 89 AND Marks=56 AND Marks = 65;
```

NOT Operator

```
SELECT StudentID,Marks,

FROM Student

WHERE NOT LastName = “Jobs”;
```

Range Operator

Range operator retrieves data based on range.

Syntax:

```
SELECT column_name1, col_name2….

FROM table_name

WHERE expression1 range_operator expression2 AND expression3
```

Types of Range operators:

BETWEEN

```
SELECT StudentID,Marks
```

```
FROM Student

WHERE Marks BETWEEN 40 AND 70;
```

NOT BETWEEN

```
SELECT FirstName,Marks,

FROM Student

WHERE Marks NOT BETWEEN 40 AND 50;
```

Retrieve Records That Match a Pattern

Data from the table can be retrieved that match a specific pattern.

The LIKE keyword matches the given character string with a specific pattern.

```
SELECT *

FROM Student

WHERE FirstName LIKE ‘Ro%’

SELECT *

FROM Student

WHERE FirstName LIKE ‘_e%’
```

Displaying in a Sequence

Use ORDER BY clause to display the data retrieved in a specific order.

```
SELECT StudentID, LastName,

FROM Student

ORDER BY Marks DESC;
```

Displaying without Duplication

The DISTINCT keyword is used to eliminate rows with duplicate values in a column.

Syntax:

```
SELECT [ALL | DISTINCT] col_names

FROM table_name

WHERE search_condition
```

```
SELECT DISTINCT Marks

FROM Student

WHERE LastName LIKE ‘o%’;
```

JOINS

Joins are used to retrieve data from more than one table together as a part of a single result set. Two or more tables can be joined based on a common attribute.

Types of JOINS:

Consider two tables Employees and EmployeeSalary

EmployeeID (PK)	FirstName	LastName	Title
1001	Ron	Brent	Developer
1002	Alex	Matt	Manager
1003	Ray	Maxi	Tester
1004	August	Berg	Quality

EmployeeID (FK)	Department	Salary
1001	Application	65000
1002	Digital Marketing	75000
1003	Web	45000
1004	Software Tools	68000

INNER JOIN

An inner join retrieves records from multiple tables by using a comparison operator on a common column.

Syntax:

```
SELECT column_name1,colomn_name2, …

FROM table1 INNER JOIN table2

ON table1.column_name = table2.column_name
```

Example:

```
SELECT e.LastName, e.Title, es.salary,

FROM e.Employees INNER JOIN es.EmployeeSalary

ON e.EmployeeID = es.EmployeeID
```

OUTER JOIN

An outer join displays the resulting set containing all the rows from one table and the matching rows from another table.

An outer join displays NULL for the column of the related table where it does not find matching records.

Syntax:

```
SELECT column_name1,colomn_name2, …

FROM table1 [LEFT|RIGHT|FULL]OUTER JOIN table2

ON table1.column_name = table2.column_name
```

Types of Outer Join

LEFT OUTER JOIN: In left outer join all rows from the table on the left side of the LEFT OUTER JOIN keyword is returned, and the matching rows from the table specified on the right side are returned the result set.

Example:

```
SELECT e.LastName, e.Title, es.salary,

FROM e.Employees LEFT OUTER JOIN es.EmployeeSalary

ON e.EmployeeID = es.EmployeeID
```

RIGHT OUTER JOIN: In right outer join all rows from the table on the right side of the RIGHT OUTER JOIN keyword are returned, and the matching rows from the table specified on the left side are returned is the result set.

Example:

```
SELECT e.LastName, e.Title, es.salary,

FROM e.Employees LEFT OUTER JOIN es.EmployeeSalary

ON e.EmployeeID = es.EmployeeID
```

FULL OUTER JOIN: It is a combination of left outer join and right outer join. This outer join returns all the matching and non-matching rows from both tables. Whilst, the matching records are displayed only once.

Example:

```
SELECT e.LastName, e.Title, es.salary,

FROM e.Employees FULL OUTER JOIN es.EmployeeSalary

ON e.EmployeeID = es.EmployeeID
```

CROSS JOIN

Also known as the Cartesian Product between two tables joins each row from one table with each row of another table. The rows in the result set is the count of rows in the first table times the count of rows in the second table.

Syntax:

```
SELECT column_name1,colomn_name2,column_name1 + column_name2 AS new_column_name

FROM table1 CROSS JOIN table2
```

EQUI JOIN

An Equi join is the same as inner join and joins tables with the help of foreign key except this join is used to display all columns from both tables.

SELF JOIN

In self join, a table is joined with itself. As a result, one row is in a table correlates with other rows in the same table. In this join, a table name is mentioned twice in the query. Hence, to differentiate the two instances of a single table, the table is given two aliases. Syntax:

```
SELECT t1.c1,t1.c2 AS column1,t2.c3,t2.c4 AS column2

FROM table1 t1 JOIN table2 t2

WHERE condition
```

Subqueries

An SQL statement that is used inside another SQL statement is termed as a subquery.

They are nested inside WHERE or HAVING clause of SELECT, INSERT, UPDATE and DELETE statements.

- Outer Query: Query that represents the parent query.
- Inner Query: Query that represents the subquery.

Using IN Keyword

If a subquery returns more than one value, we might execute the outer query if the values within the columns specified in the condition match any value in the result set of the subquery.

Syntax:

```
SELECT column, column
```

```
FROM table_name

WHERE column [NOT] IN

(SELECT column

 FROM table_name [WHERE conditional_expression] )
```

Using EXISTS Keyword

EXISTS clause is used with subquery to check if a set of records exists.

TRUE value is returned by the subquery in case if the subquery returns any row.

Syntax:

```
SELECT column, column

FROM table_name

WHERE EXISTS

(SELECT column_name FROM table_name WHERE condition)
```

Using Nested Subqueries

A subquery can contain more than one subqueries. Subqueries are used when the condition of a query is dependent on the result of another query, which is, in turn, is dependent on the result of another subquery.

Syntax:

```
SELECT column, column

FROM table_name

WHERE column_name expression_operator

(SELECT column_list FROM table_name

 WHERE column_name expression_operator

 (SELECT column_list FROM table_name

 WHERE [condition] ) )
```

Correlated Subquery

A correlated subquery can be defined as a query that depends on the outer query for its evaluation.

Using Functions to Customize ResultSet

Various in-built functions can be used to customize the result set.

Syntax:

```
SELECT function_name (parameters)
```

Using String Functions

String values in the result set can be manipulated by using string functions.

They are used with char and varchar data types.

Following are the commonly used string functions are:

Function Name	Example
left	`SELECT left ('RICHARD' ,4)`
len	`SELECT len ('RICHARD')`
lower	`SELECT lower ('RICHARD')`
reverse	`SELECT reverse ('ACTION')`
right	`SELECT right ('RICHARD' ,4)`

space	`SELECT 'RICHARD' + space(2) + 'HILL'`
str	`SELECT str (123.45,6,2)`
substring	`SELECT substring ('Weather' ,2,2)`
upper	`SELECT upper ('RICHARD')`

Using Date Functions

Date functions are used to manipulate date time values or to parse the date values.

Date parsing includes extracting components, such as day, month, and year from a date value.

Some of the commonly used date functions are:

Function Name	Parameters	Description
dateadd	(date part, number, date)	Adds the number of date parts to the date.
datediff	(date part, date1, date2)	Calculates the number of date parts between two dates.
Datename	(date part, date)	Returns date part from the listed as a character value.
datepart	(date part, date)	Returns date part from the listed as an integer.
getdate	0	Returns current date and time
day	(date)	Returns an integer, which represents the day.
month	(date)	Returns an integer, which represents the month.

year	(date)	Returns an integer, which represents the year.

Using Mathematical Functions

Numeric values in a result set can be manipulated in using mathematical functions.

The following table lists the mathematical functions:

Function Name	Parameters	Description
abs	(numeric_expression)	Returns an absolute value
acts,asin,atan	(float_expression)	Returns an angle in radians
cos, sin, cot,tan	(float_expression)	Returns the cosine, sine, cotangent, or tangent of the angle in radians.
degrees	(numeric_expression)	Returns the smallest integer greater than or equal to specifies value.
exp	(float_expression)	Returns the exponential value of the specified value.
floor	(numeric_expression)	Returns the largest integer less than or equal to the specified value.
log	(float_expression)	Returns the natural logarithm of the specified value.
pi	0	Returns the constant value of 3.141592653589793
power	(numeric_expression,y)	Returns the value of numeric expression to the value of y
radians	(numeric_expression)	Converts from degrees to radians.
rand	([seed])	Returns a random float number between 0 and 1.
round	(numeric_expression,length)	Returns a numeric expression rounded off to the length specified as an integer expression.
sign	(numeric_expression)	Returns positive, negative or zero.
sqrt	(float_expression)	Returns the square root of the specified value.

Using Ranking Functions

Ranking functions are used to generate sequential numbers for each row to give a rank based on specific criteria.

Ranking functions return a ranking value for each row. Following functions are used to rank the records:

- row_number Function: This function returns the sequential numbers, starting at 1, for the rows in a result set based on a column.
- rank Function: This function returns the rank of each row in a result set based on specified criteria.
- dense_rank Function: The dense_rank() function is used where consecutive ranking values need to be given based on specified criteria.

These functions use the OVER clause that determines the ascending or descending sequence in which rows are assigned a rank.

Using Aggregate Functions

The aggregate functions, on execution, summarize the values for a column or group of columns and produce a single value.

Syntax:

```
SELECT aggrgate_function ([ALL | DISTINCT] expression)

FROM table_name
```

Following are the aggregate functions:

Function Name	Description
avg	returns the average of values in a numeric expression, either all or distinct.
count	returns the number of values in an expression, either all or distinct.
min	returns the lowest value in an expression.
max	returns the highest value in an expression.
sum	returns the total of values in an expression, either all or distinct.

GROUPING DATA

Grouping data means to view data that match a specific criteria to be displayed together in the result set.

Data can be grouped by using GROUP BY, COMPUTE,COMPUTE BY and PIVOT clause in the SELECT statement.

GROUP BY Clause

Summarizes the result set into groups as defined in the query by using aggregate functions.

Syntax:

```
SELECT column_list

FROM table_name

WHERE condition

[GROUP BY [ALL] expression]

[HAVING search_condition]
```

COMPUTE and COMPUTE BY Clause

This COMPUTE clause, with the SELECT statement, is used to generate summary rows by using aggregate functions in the query result.

The COMPUTE BY clause can be used to calculate summary values of the result set on a group of data.

Syntax:

```
SELECT column_list

FROM table_name

ORDER BY column_name

COMPUTE aggregate_function (column_name)

[BY column_name]
```

PIVOT Clause

The PIVOT operator is used to transform a set of columns into values, PIVOT rotates a table-valued expression by turning the unique values from one column in the expression into multiple columns in the output.

Syntax:

```
SELECT *

FROM table_name

PIVOT (aggregate function (value_column)

 FOR pivot_column

 IN (column_list)

) table_alias
```