Warehouse Managment System

Database Management System

MSCS 542L-256-24F

TechBoys

Marist College
School of Computer Science and Mathematics

Submitted To:
Dr. Reza Sadeghi

Sept 18, 2024

PROJECT REPORT OF WAREHOUSE MANAGEMENT SYSTEM

Team Name
TechBoys

Team Members

Sumanth Kumar Katapally	SumanthKumar.Katapally1@marist.edu (Team Head)
Abhijeet Cherungottil	Abhijeet.Cherungottil1@marist.edu (Team Member)
Sagar Shankaran	Sagar.Shankaran1@marist.edu (Team Member)

TABLE OF CONTENTS

1. DESCRIPTION OF TEAM MEMBERS

Sumanth Kumar Katapally:

I attained my Bachelor's degree in Computer Science from Keshav Memorial Institute of Technology in 2022. I kickstarted my career as a Software Developer Intern at Virtusa, after which I took up a full-time role of a Software Developer at DBS TECH thereby gaining an overall experience of 2 years and 5 months. I am proficient in Java, Spring, Spring Boot, Angular, MongoDB and MySQL. I came to Marist College to pursue my MS in Computer Science concentrating in Artificial Intelligence.

Abhijeet Cherungottil:

I am a passionate computer science graduate currently pursuing an MS in Cloud Computing at Marist College, set to graduate in the 2026 batch. I also have 3 years of hands-on experience in iPhone app development, with expertise in Swift 5 and Swift UI. My current team is from India, which creates familiarity within the group and allows us to communicate easily with each other. I chose Sumanth because he mentioned having previous experience with GitHub and actively volunteered.

Sagar Shankaran:

I am a 2026 batch student of Masters in AI program at Marist College. I have experience in full stack development for 3 years. I have worked across various projects including MERN stack and Android Apps. I have chosen this team because all the members have a good understanding of each other and have profound knowledge of the project. Abhijeet and Sumanth have good relevant skills.

2. GITHUB REPOSITORY

https://github.com/Sumanthkatapally/DBMS_PROJECT.git

3. PRESENTATION SUBJECT

Data Types specify the type of data that a column can contain. String Data Types specify that the column can hold data made up of characters.[1][2]

String Data Types:

CHAR: Creates a fixed length that one can declare after creating the table [3]

- This length value can be between 0 and 255
- If one skips declaring the length variable, it is automatically assigned to 1 [4]
- If the user enters a string that is less than the declared value, the database will pad
- the remaining space to fill it to the correct value [3]
- Ex: char(24) can hold up to 24 characters. If the user enters only 20 characters, the
- remaining 4 spaces will be padded.

VARCHAR: Allows a varying length for the inputted string [2]

- Length ranges from 0 to 65,535
- VARCHAR inputs are not padded: varchar(24) would only specify that the column can hold a maximum of 24 characters. 20 characters would be stored as 20 characters, not 24 (20 + 4 padding)
- This can reduce the amount of storage required to save data

BINARY: Stores a set string of bytes.[4]

- Similar to CHAR, except stores binary strings instead of nonbinary strings

VARBINARY: Variable length binary data.[4]

- Similar to VARCHAR, except stores binary strings instead of nonbinary strings
- Unlike CHAR and VARCHAR, the length of BINARY and VARBINARY is
- measured in byes but not in characters

BLOB: A binary large object that can hold a variable amount of data[5]

- Four blob types: tinyblob, blob, mediumblob and longblob
- They differ only in the maximum length of values they can hold
- Blob values are treated as binary strings(byte strings), they have binary character

TEXT: Stores any kind of text data. Can contain both single-byte and multibyte characters

- Text values are treated as nonbinary strings (character strings), they have a character set other than binary

ENUM: A string object with a value chosen from a list of permitted values that are [6]

enumerated explicitly in the column specification table at creation time

- The strings you specify as input values are automatically encoded as numbers.
- The numbers are translated back to the corresponding strings in query results
- Enum can have a maximum of 65,535 distinct elements

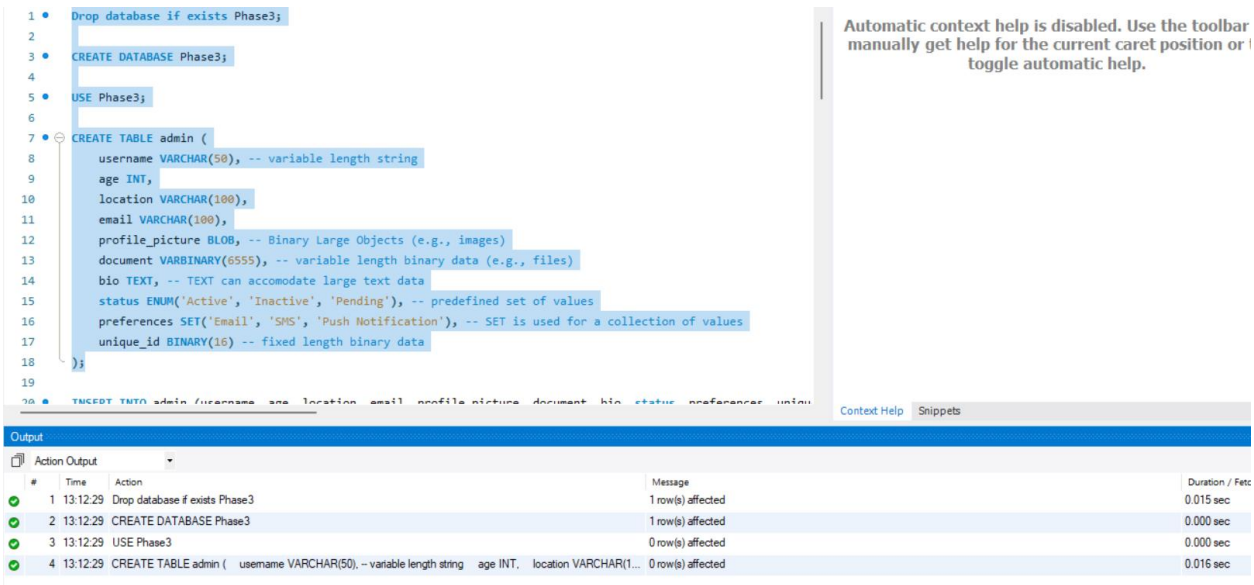
• **SET:** A string object that can have zero or more values each of which must be chosen from a list of permitted values specified when the table is created [7]

- Set column values that consist of multiple set members must be separated by

- columns
- A set column can have a maximum of 64 distinct members
- For example, a column specified as SET('Email', ‘SMS’) NOT NULL can have any of
- the values '"Email' 'SMS' 'Email,SMS'"

String Data Type Examples:

Table creation and Insertion:



```
Drop database if exists Phase3;

CREATE DATABASE Phase3;

USE Phase3;

CREATE TABLE admin (
    username VARCHAR(50), -- variable length string
    age INT,
    location VARCHAR(100),
    email VARCHAR(100),
    profile_picture BLOB, -- Binary Large Objects (e.g., images)
    document VARBINARY(6555), -- variable length binary data (e.g., files)
    bio TEXT, -- TEXT can accomodate large text data
    status ENUM('Active', 'Inactive', 'Pending'), -- predefined set of values
    preferences SET('Email', 'SMS', 'Push Notification'), -- SET is used for a collection of values
    unique_id BINARY(16) -- fixed length binary data
);
```

Automatic context help is disabled. Use the toolbar manually get help for the current caret position or toggle automatic help.

Context Help | Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetc
1	13:12:29	Drop database if exists Phase3	1 row(s) affected	0.015 sec
2	13:12:29	CREATE DATABASE Phase3	1 row(s) affected	0.000 sec
3	13:12:29	USE Phase3	0 row(s) affected	0.000 sec
4	13:12:29	CREATE TABLE admin (username VARCHAR(50), -- variable length string age INT, location VARCHAR(1...	0 row(s) affected	0.016 sec

```
INSERT INTO admin (username, age, location, email, profile_picture, document, bio, status, preferences, uniqu
VALUES (
    'Sumanth',
    24,
    'New York',
    'sumanth@gmail.com',
    '0x89504E470D0A1A0A0000000D494844520000001000000010080600000001F4FBD61', -- Example BLOB data
    '0x255044462D312E350D0A25E2E3CFD30D0A312030206F626A0D0A3C3C2F4C656E6774682034', -- Example VARBINARY data
    'Intro of sumanth', -- Example TEXT data
    'Active', -- Status ENUM
    'Email,SMS', -- Preferences SET
    UNHEX('4F6A1F8E9D6F4E8B8C9A8F2C5D4E6F7D') -- Example BINARY data
);

INSERT INTO admin (username, age, location, email, profile_picture, document, bio, status, preferences, uniqu
VALUES (
    'Abhijeet',
    24,
```

Automatic context help is disabled. Use the toolbar manually get help for the current caret position or t toggle automatic help.

Context Help | Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetc
1	13:13:36	INSERT INTO admin (username, age, location, email, profile_picture, document, bio, status, preferences, unique_i...	1 row(s) affected	0.016 sec
2	13:13:36	INSERT INTO admin (username, age, location, email, profile_picture, document, bio, status, preferences, unique_i...	1 row(s) affected	0.000 sec
3	13:13:36	INSERT INTO admin (username, age, location, email, profile_picture, document, bio, status, preferences, unique_i...	1 row(s) affected	0.000 sec

String Functions:[9]

<u>LENGTH ( ) :</u> Provides length of string

```
SELECT username, LENGTH(username) AS username_length FROM admin; -- Length of string
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

username	username_length
Sumanth	7
Abhijeet	8
Sagar	5

Result Grid | Form Editor | Field Types

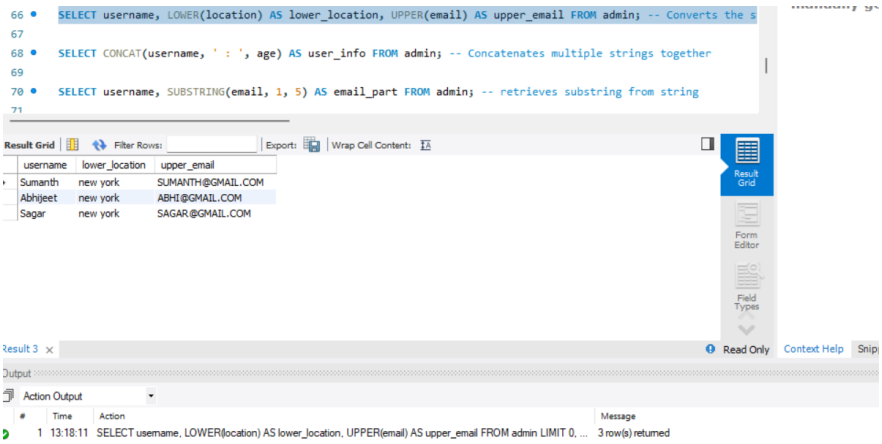
Result 2 | Read Only | Context Help | Snippets

Output

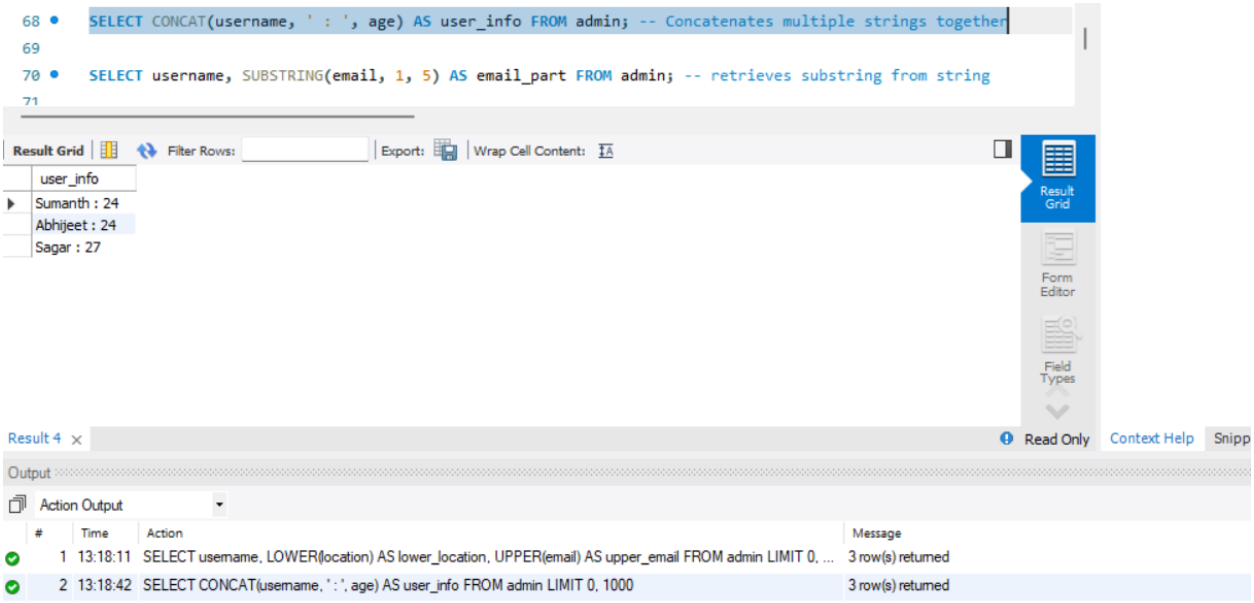
Action Output

#	Time	Action	Message	Duration / Fetch
1	13:17:16	SELECT username, LENGTH(username) AS username_length FROM admin LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000

Lower () & Upper (): Converts the string to lower and upper case



Concat: Concatenates multiple strings together



Substring (): Retrieves substring from string

```
SELECT username, SUBSTRING(email, 1, 5) AS email_part FROM admin; -- retrieves substring from string

SELECT username, INSTR(email, 'gmail') AS gmail_position FROM admin; -- finds the position of the first occur

SELECT username, REPLACE(location, 'New York', 'NY') AS short_location FROM admin; -- replacing string
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

username	email_part
Sumanth	suman
Abhijeet	abhi@
Sagar	sagar

Result 5 × Read Only

Output

Action Output

#	Time	Action	Message
1	13:18:11	SELECT username, LOWER(location) AS lower_location, UPPER(email) AS upper_email FROM admin LIMIT 0, ...	3 row(s) returned
2	13:18:42	SELECT CONCAT(username, ' : ', age) AS user_info FROM admin LIMIT 0, 1000	3 row(s) returned
3	13:19:05	SELECT username, SUBSTRING(email, 1, 5) AS email_part FROM admin LIMIT 0, 1000	3 row(s) returned

Replace (): Replaces string with provided input string.

```
SELECT username, REPLACE(location, 'New York', 'NY') AS short_location FROM admin; -- replacing string

SELECT username, TRIM(both ' ' FROM bio) AS trimmed_bio FROM admin; -- removes spaces from the string
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

username	short_location
Sumanth	NY
Abhijeet	NY
Sagar	NY

Result 6 × Read Only Context Help Snipp

Output

Action Output

#	Time	Action	Message
1	13:18:11	SELECT username, LOWER(location) AS lower_location, UPPER(email) AS upper_email FROM admin LIMIT 0, ...	3 row(s) returned
2	13:18:42	SELECT CONCAT(username, ' : ', age) AS user_info FROM admin LIMIT 0, 1000	3 row(s) returned
3	13:19:05	SELECT username, SUBSTRING(email, 1, 5) AS email_part FROM admin LIMIT 0, 1000	3 row(s) returned
4	13:19:34	SELECT username, REPLACE(location, 'New York', 'NY') AS short_location FROM admin LIMIT 0, 1000	3 row(s) returned

TRIM (): Removes spaces from the string

```
76 • SELECT username, TRIM(both ' ' FROM bio) AS trimmed_bio FROM admin; -- removes spaces from the string
77
78
79 -- Search
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

username	trimmed_bio
Sumanth	Intro of sumanth
Abhijeet	Intro of Abhijeet
Sagar	Intro of Sagar

Result 7 × Read Only Context Help Snip

Output

Action Output

#	Time	Action	Message
1	13:18:11	SELECT username, LOWER(location) AS lower_location, UPPER(email) AS upper_email FROM admin LIMIT 0, ...	3 row(s) returned
2	13:18:42	SELECT CONCAT(username, ' : ', age) AS user_info FROM admin LIMIT 0, 1000	3 row(s) returned
3	13:19:05	SELECT username, SUBSTRING(email, 1, 5) AS email_part FROM admin LIMIT 0, 1000	3 row(s) returned
4	13:19:34	SELECT username, REPLACE(location, 'New York', 'NY') AS short_location FROM admin LIMIT 0, 1000	3 row(s) returned
5	13:19:49	SELECT username, TRIM(both ' ' FROM bio) AS trimmed_bio FROM admin LIMIT 0, 1000	3 row(s) returned

Search:

```
81 • SELECT * FROM admin WHERE email LIKE '%gmail%';
82
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

username	age	location	email	profile_picture	document	bio	status	preferences	unique_id
Sumanth	24	New York	sumanth@gmail.com	BLOB	BLOB	Intro of sumanth	Active	Email,SMS	BLOB
Abhijeet	24	New York	abhi@gmail.com	BLOB	BLOB	Intro of Abhijeet	Pending	Email	BLOB
Sagar	27	New York	sagar@gmail.com	BLOB	BLOB	Intro of Sagar	Active	SMS	BLOB

admin 8 × Read Only Context H

Output

Action Output

#	Time	Action	Message
1	13:18:11	SELECT username, LOWER(location) AS lower_location, UPPER(email) AS upper_email FROM admin LIMIT 0, ...	3 row(s) returned
2	13:18:42	SELECT CONCAT(username, ' : ', age) AS user_info FROM admin LIMIT 0, 1000	3 row(s) returned
3	13:19:05	SELECT username, SUBSTRING(email, 1, 5) AS email_part FROM admin LIMIT 0, 1000	3 row(s) returned
4	13:19:34	SELECT username, REPLACE(location, 'New York', 'NY') AS short_location FROM admin LIMIT 0, 1000	3 row(s) returned
5	13:19:49	SELECT username, TRIM(both ' ' FROM bio) AS trimmed_bio FROM admin LIMIT 0, 1000	3 row(s) returned
6	13:20:20	SELECT * FROM admin WHERE email LIKE '%gmail%' LIMIT 0, 1000	3 row(s) returned

```
83 •  SELECT * FROM admin WHERE FIND_IN_SET('Email', preferences) > 0;
84
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

username	age	location	email	profile_picture	document	bio	status	preferences	unique_id
Sumanth	24	New York	sumanth@gmail.com	BLOB	BLOB	Intro of sumanth	Active	Email,SMS	BLOB
Abhijeet	24	New York	abhi@gmail.com	BLOB	BLOB	Intro of Abhijeet	Pending	Email	BLOB

admin 9 ×

Output

Action Output

#	Time	Action	Message
2	13:18:42	SELECT CONCAT(username, ' : ', age) AS user_info FROM admin LIMIT 0, 1000	3 row(s) returned
3	13:19:05	SELECT username, SUBSTRING(email, 1, 5) AS email_part FROM admin LIMIT 0, 1000	3 row(s) returned
4	13:19:34	SELECT username, REPLACE(location, 'New York', 'NY') AS short_location FROM admin LIMIT 0, 1000	3 row(s) returned
5	13:19:49	SELECT username, TRIM(both ' ' FROM bio) AS trimmed_bio FROM admin LIMIT 0, 1000	3 row(s) returned
6	13:20:20	SELECT * FROM admin WHERE email LIKE '%gmail%' LIMIT 0, 1000	3 row(s) returned
7	13:21:01	SELECT * FROM admin WHERE FIND_IN_SET('Email', preferences) > 0 LIMIT 0, 1000	2 row(s) returned

JSON Data Types:[8]

JSON is a lightweight, text-based data format designed to be easy for humans to read and write, and easy for machines to parse and generate.

Introduced in MySQL 5.7.8 for better support of structured and unstructured data.

Often used for APIs and storing complex data like arrays, objects, etc.

Creation & Insertion:

Storing structured data as a JSON column allows you to store complex data types.

Create and insertion of a data to table with a JSON column

```
-- Inserting JSON data type values into the table

-- Direct Insertion of JSON object
INSERT INTO json_demo_table(json_data, description)
VALUES ('{"key1":"value1","key2":"value2"}', 'Direct JSON object insertion');


INSERT INTO json_demo_table(json_data, description)
VALUES ('[3, "string1"]', 'Direct JSON array insertion');

-- JSON object insertion using JSON_OBJECT() function
INSERT INTO json_demo_table(json_data, description)
VALUES (JSON_OBJECT("key3", "value3", "key4", "value4"), 'Insertion using JSON_OBJECT()');
```

Output

Action Output

#	Time	Action	Message
1	14:40:22	CREATE TABLE json_demo_table (json_data JSON, description VARCHAR(100))	0 row(s) affected
2	14:40:57	INSERT INTO json_demo_table(json_data, description) VALUES ('{"key1":"value1","key2":"value2"}', 'Direct J...	1 row(s) affected
3	14:40:57	INSERT INTO json_demo_table(json_data, description) VALUES ('[3, "string1"]', 'Direct JSON array insertion')	1 row(s) affected

Insertion using functions

```sql
-- JSON object insertion using JSON_OBJECT() function
INSERT INTO json_demo_table(json_data, description)
VALUES (JSON_OBJECT("key3", "value3", "key4", "value4"), 'Insertion using JSON_OBJECT()');

-- JSON array insertion using JSON_ARRAY() function
INSERT INTO json_demo_table(json_data, description)
VALUES (JSON_ARRAY("string2", 3, '{"key5":"value5","key6":"value6"}'), 'Insertion using JSON_ARRAY()');

-- Array inside JSON object
INSERT INTO json_demo_table(json_data, description)
VALUES ('{"key1":["string3", 6], "key2":"value2"}', 'Nested JSON: array in object');

-- Display the contents of the table
SELECT * FROM json_demo_table;


-- Merging JSON documents

SELECT JSON_MERGE_PRESERVE(json_data, '[1, 2]'), JSON_MERGE_PRESERVE('[1, 2]', json_data), JSON_MERGE_PRESERVE(json_data, '{
```

Output

Action Output

#	Time	Action	Message
1	14:40:22	CREATE TABLE json_demo_table (json_data JSON, description VARCHAR(100))	0 row(s) affected
2	14:40:57	INSERT INTO json_demo_table(json_data, description) VALUES ('{"key1":"value1","key2":"value2"}', 'Direct J...	1 row(s) affected
3	14:40:57	INSERT INTO json_demo_table(json_data, description) VALUES ('[3, "string1"]', 'Direct JSON array insertion')	1 row(s) affected
4	14:41:43	INSERT INTO json_demo_table(json_data, description) VALUES (JSON_OBJECT("key3", "value3", "key4", "va...	1 row(s) affected
5	14:41:44	INSERT INTO json_demo_table(json_data, description) VALUES (JSON_ARRAY("string2", 3, '{"key5":"value5",...	1 row(s) affected

Using array in JSON object:

```sql
-- Array inside JSON object
INSERT INTO json_demo_table(json_data, description)
VALUES ('{"key1":["string3", 6], "key2":"value2"}', 'Nested JSON: array in object');

-- Display the contents of the table
SELECT * FROM json_demo_table;
```

Output

Action Output

#	Time	Action	Message
2	14:40:57	INSERT INTO json_demo_table(json_data, description) VALUES ('{"key1":"value1","key2":"value2"}', 'Direct...	1 row(s) affected
3	14:40:57	INSERT INTO json_demo_table(json_data, description) VALUES ('[3, "string1"]', 'Direct JSON array insertion')	1 row(s) affected
4	14:41:43	INSERT INTO json_demo_table(json_data, description) VALUES (JSON_OBJECT("key3", "value3", "key4", ...	1 row(s) affected
5	14:41:44	INSERT INTO json_demo_table(json_data, description) VALUES (JSON_ARRAY("string2", 3, '{"key5":"value...	1 row(s) affected
6	14:42:12	INSERT INTO json_demo_table(json_data, description) VALUES ('{"key1":["string3", 6], "key2":"value2"}', 'N...	1 row(s) affected

```
35    -- Display the contents of the table
36 •  SELECT * FROM json_demo_table;
37
38
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	json_data	description
▶	{"key1": "value1", "key2": "value2"}	Direct JSON object insertion
	[3, "string1"]	Direct JSON array insertion
	{"key3": "value3", "key4": "value4"}	Insertion using JSON_OBJECT()
	["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v...	Insertion using JSON_ARRAY()
	{"key1": ["string3", 6], "key2": "value2"}	Nested JSON: array in object

json_demo_table 10 ×

Output

Action Output

#	Time	Action	Message
2	14:40:57	INSERT INTO json_demo_table(json_data, description) VALUES ('{"key1":"value1","key2":"value2"}', 'Direct...	1 row(s) affected
3	14:40:57	INSERT INTO json_demo_table(json_data, description) VALUES ('[3, "string1"]', 'Direct JSON array insertion')	1 row(s) affected
4	14:41:43	INSERT INTO json_demo_table(json_data, description) VALUES (JSON_OBJECT("key3", "value3", "key4", ...	1 row(s) affected
5	14:41:44	INSERT INTO json_demo_table(json_data, description) VALUES (JSON_ARRAY("string2", 3, '{"key5":"value...	1 row(s) affected
6	14:42:12	INSERT INTO json_demo_table(json_data, description) VALUES ('{"key1":["string3", 6], "key2":"value2"}', 'N...	1 row(s) affected
7	14:42:23	SELECT * FROM json_demo_table LIMIT 0, 1000	5 row(s) returned

Merge:

<u>JSON_MERGE_PRESERVE( ):</u> Merging JSON data type columns with JSON array and vice versa

<u>JSON_MERGE_PATCH( ):</u> Merging JSON data type columns with JSON array and vice versa

```
-- Merging JSON documents

SELECT JSON_MERGE_PRESERVE(json_data, '[1, 2]'),JSON_MERGE_PRESERVE('[1, 2]', json_data),JSON_MERGE_PRESERVE(json_data, '{"keyme
JSON_MERGE_PATCH(json_data, '[1, 2]'),JSON_MERGE_PATCH('[1, 2]', json_data),JSON_MERGE_PATCH(json_data, '{"keymerge":"valuemerge
FROM json_demo_table;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ta,	JSON_MERGE_PRESERVE('{"keymerge":"valuemer(json_data)	JSON_MERGE_PATCH(json_data, '[1, 2]')	JSON_MERGE_PATCH('[1, 2]', json_data)	JSON_MERGE_PATCH(json_ '{"keymerge":"valuemerge"}
, "keymerge"...	{"key1": "value1", "key2": "value2", "keymerge"...	[1, 2]	{"key1": "value1", "key2": "value2"}	{"key1": "value1", "key2": "v
nerge"}]	[{"keymerge": "valuemerge"}, 3, "string1"]	[1, 2]	[3, "string1"]	{"keymerge": "valuemerge"}
, "keymerge"...	{"key3": "value3", "key4": "value4", "keymerge"...	[1, 2]	{"key3": "value3", "key4": "value4"}	{"key3": "value3", "key4": "v
\"key6\":\"v...	[{"keymerge": "valuemerge"}, "string2", 3, "{\"k...	[1, 2]	["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v...	{"keymerge": "valuemerge"}
ue2", "keym...	{"key1": ["string3", 6], "key2": "value2", "keym...	[1, 2]	{"key1": ["string3", 6], "key2": "value2"}	{"key1": ["string3", 6], "key2

Result 11

Output

Action Output

#	Time	Action	Message
3	14:40:57	INSERT INTO json_demo_table(json_data, description) VALUES ('[3, "string1"]', 'Direct JSON array insertion')	1 row(s) affected
4	14:41:43	INSERT INTO json_demo_table(json_data, description) VALUES (JSON_OBJECT("key3", "value3", "key4", ...	1 row(s) affected
5	14:41:44	INSERT INTO json_demo_table(json_data, description) VALUES (JSON_ARRAY("string2", 3, '{"key5":"value...	1 row(s) affected
6	14:42:12	INSERT INTO json_demo_table(json_data, description) VALUES ('{"key1":["string3", 6], "key2":"value2"}', 'N...	1 row(s) affected
7	14:42:23	SELECT * FROM json_demo_table LIMIT 0, 1000	5 row(s) returned
8	14:43:38	SELECT JSON_MERGE_PRESERVE(json_data, '[1, 2]'),JSON_MERGE_PRESERVE('[1, 2]', json_data),JSO...	5 row(s) returned

JSON Functions:[8]

<u>JSON_ARRAY_APPEND ():</u>

```
-- Appending integer element into an array at index 1
SELECT json_data, JSON_ARRAY_APPEND(json_data, '$[1]', 1)
FROM json_demo_table;

-- Inserting integer element into an array at index 0
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

json_data	JSON_ARRAY_APPEND(json_data, '$[1]', 1)
{"key1": "value1", "key2": "value2"}	{"key1": "value1", "key2": "value2"}
[3, "string1"]	[3, ["string1", 1]]
{"key3": "value3", "key4": "value4"}	{"key3": "value3", "key4": "value4"}
["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v...	["string2", [3, 1], "{\"key5\":\"value5\",\"key6\"...
{"key1": ["string3", 6], "key2": "value2"}	{"key1": ["string3", 6], "key2": "value2"}

JSON_ARRAY_INSERT():

```
-- Inserting integer element into an array at index 0
SELECT JSON_ARRAY_INSERT(json_data, '$[0]', 1)
FROM json_demo_table;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

JSON_ARRAY_INSERT(json_data, '$[0]', 1)
{"key1": "value1", "key2": "value2"}
[1, 3, "string1"]
{"key3": "value3", "key4": "value4"}
[1, "string2", 3, "{\"key5\":\"value5\",\"key6\":\...
{"key1": ["string3", 6], "key2": "value2"}

Result 13

Output

JSON_STORAGE_SIZE ():

```
-- Calculating the storage size of JSON objects
SELECT json_data, JSON_STORAGE_SIZE(json_data)
FROM json_demo_table;

-- Determining the type of JSON elements
SELECT json_data, JSON_TYPE(json_data)
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

json_data	JSON_STORAGE_SIZE(json_data)
{"key1": "value1", "key2": "value2"}	41
[3, "string1"]	19
{"key3": "value3", "key4": "value4"}	41
["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v...	56
{"key1": ["string3", 6], "key2": "value2"}	52

JSON_TYPE ():

```
-- Determining the type of JSON elements
SELECT json_data, JSON_TYPE(json_data)
FROM json_demo_table;

-- Getting the length of JSON elements (array length or number of key-value pair
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

json_data	JSON_TYPE(json_data)
{"key1": "value1", "key2": "value2"}	OBJECT
[3, "string1"]	ARRAY
{"key3": "value3", "key4": "value4"}	OBJECT
["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v...	ARRAY
{"key1": ["string3", 6], "key2": "value2"}	OBJECT

Result 15 ×

Output

JSON_LENGTH():

```
-- Getting the length of JSON elements (array length or number of key-value pairs in object)
SELECT json_data, JSON_LENGTH(json_data)
FROM json_demo_table;

-- Validating JSON objects (returns 1 if valid, 0 otherwise)
SELECT JSON_VALID('string'), JSON_VALID('"string1"');
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

json_data	JSON_LENGTH(json_data)
{"key1": "value1", "key2": "value2"}	2
[3, "string1"]	2
{"key3": "value3", "key4": "value4"}	2
["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v...	3
{"key1": ["string3", 6], "key2": "value2"}	2

Result 16 ×

Output

Action Output

JSON_VALID: Validating JSON objects (returns 1 if valid, 0 otherwise)

```
-- Validating JSON objects (returns 1 if valid, 0 otherwise)
SELECT JSON_VALID('string'), JSON_VALID('"string1"');

-- Search and selection within JSON documents

-- Using JSON_EXTRACT() to search for values by key
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	JSON_VALID('string')	JSON_VALID('"string1"')
▶	0	1

JSON_INSERT:

```
-- JSON_INSERT(): Inserting a new key-value pair into the JSON object
SELECT
    json_data,
    JSON_INSERT(json_data, '$.keynew', 'newval')
FROM json_demo_table;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	json_data	JSON_INSERT(json_data, '$.keynew', 'newval')
▶	{"key1": "value1", "key2": "value2"}	{"key1": "value1", "key2": "value2", "keynew": ...
	[3, "string1"]	[3, "string1"]
	{"key3": "value3", "key4": "value4"}	{"key3": "value3", "key4": "value4", "keynew": ...
	["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v...	["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v...
	{"key1": ["string3", 6], "key2": "value2"}	{"key1": ["string3", 6], "key2": "value2", "keyne...

JSON_REMOVE:

```
-- JSON_REMOVE(): Removing an element from an array or object
SELECT
    json_data,
    JSON_REMOVE(json_data, '$[0]')  -- Remove the first element of the array
FROM json_demo_table;

-- JSON_REPLACE(): Replacing the value of a key in the JSON object
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

json_data	JSON_REMOVE(json_data, '$[0]')
{"key1": "value1", "key2": "value2"}	{"key1": "value1", "key2": "value2"}
[3, "string1"]	["string1"]
{"key3": "value3", "key4": "value4"}	{"key3": "value3", "key4": "value4"}
["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v...	[3, "{\"key5\":\"value5\",\"key6\":\"value6\"}"]
{"key1": ["string3", 6], "key2": "value2"}	{"key1": ["string3", 6], "key2": "value2"}

JSON_REPLACE:

```
-- JSON_REPLACE(): Replacing the value of a key in the JSON object
SELECT
    json_data,
    JSON_REPLACE(json_data, '$.key1', 'val3')
FROM json_demo_table;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

json_data	JSON_REPLACE(json_data, '$.key1', 'val3')
{"key1": "value1", "key2": "value2"}	{"key1": "val3", "key2": "value2"}
[3, "string1"]	[3, "string1"]
{"key3": "value3", "key4": "value4"}	{"key3": "value3", "key4": "value4"}
["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v...	["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v...
{"key1": ["string3", 6], "key2": "value2"}	{"key1": "val3", "key2": "value2"}

JSON_SET:

```
-- JSON_SET(): Set object with key 'key1' as 'val3'
SELECT
    json_data,
    JSON_SET(json_data, '$.key1', 'val3')
FROM json_demo_table;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	json_data	JSON_SET(json_data, '$.key1', 'val3')
▶	{"key1": "value1", "key2": "value2"}	{"key1": "val3", "key2": "value2"}
	[3, "string1"]	[3, "string1"]
	{"key3": "value3", "key4": "value4"}	{"key1": "val3", "key3": "value3", "key4": "valu...
	["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v...	["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v...
	{"key1": ["string3", 6], "key2": "value2"}	{"key1": "val3", "key2": "value2"}

JSON_EXTRACT:

```
-- Using JSON_EXTRACT() to search for values by key
SELECT
    json_data,
    JSON_EXTRACT(json_data, '$.key1'),
    JSON_EXTRACT(json_data, '$[0].key1')  -- arrays with objects at index 0
FROM json_demo_table;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	json_data	JSON_EXTRACT(json_data, '$.key1')	JSON_EXTRACT(json_data, '$[0].key1')
▶	{"key1": "value1", "key2": "value2"}	"value1"	"value1"
	[3, "string1"]	NULL	NULL
	{"key3": "value3", "key4": "value4"}	NULL	NULL
	["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v...	NULL	NULL
	{"key1": ["string3", 6], "key2": "value2"}	["string3", 6]	["string3", 6]

4. REFERENCES

[1] SQL data types

[2] String type syntax

[3] Char

[4] Binary and Varbinary

[5] Blob

[6] Enum

[7] Set

[8] JSON data types

[9] String Functions