JSON in SQL

Create a table with a primary-key column and a column of JSON data type only from version 12C.

```sql
CREATE TABLE j_purchaseorder (id VARCHAR2 (32) NOT NULL PRIMARY KEY,

date_loaded TIMESTAMP (6) WITH TIME ZONE, po_document JSON);
```

For Oracle 11g: Create a Table with a Primary Key + JSON Stored in CLOB

```sql
CREATE TABLE user_profiles ( id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY, profile_data CLOB);
```

- **id NUMBER GENERATED ALWAYS AS IDENTITY**

Acts like an auto-incrementing primary key.

Oracle 11g doesn't include `IS JSON` constraints (introduced in 12c), so you must validate JSON manually—typically via:

1. A **trigger**
2. A **PL/SQL function** that checks JSON validity

Solution is to create auto increment - Create a Sequence for Auto-Incrementing IDs

```sql
CREATE SEQUENCE user_profiles_seq START WITH 1 INCREMENT BY 1;
```

Create a Trigger to Auto-Fill the ID on INSERT

```sql
CREATE OR REPLACE TRIGGER user_profiles_trg

BEFORE INSERT ON user_profiles

FOR EACH ROW

BEGIN

  IF :NEW.id IS NULL THEN

    SELECT user_profiles_seq.NEXTVAL INTO :NEW.id FROM dual;

  END IF;

END;

/
```

Simple Insert JSON Data into the Table:

```sql
INSERT INTO user_profiles (profile_data) VALUES
('{"name":"Alice","age":30,"likes":["music","coffee"]}');
```

```
INSERT INTO user_profiles (profile_data) VALUES ('{"name":"Bob","active":true}');

INSERT INTO user_profiles (profile_data) VALUES
('{"name":"Charlie","roles":["admin","editor"]}');
```

Verify Inserts

```
SELECT * FROM user_profiles;
```

Nested JSON object data into the Table:

```
INSERT INTO user_profiles (profile_data)

VALUES ('{"name":"Bob","address":{"city":"London","zip":22045}}');
```

Array inside JSON object data into the Table:

```
INSERT INTO user_profiles (profile_data)

VALUES ('{"name":"Charlie","skills":["SQL","Java","Python"]}');
```

Mixed types JSON object data into the Table:

```
INSERT INTO user_profiles (profile_data)

VALUES ('{"name":"Diana","active":true,"score":89.5}');
```

Large JSON stored in CLOB

```
INSERT INTO user_profiles (profile_data)

VALUES ('{

  "name":"Edward",

  "devices":[

    {"type":"mobile","os":"android"},

    {"type":"laptop","os":"windows"}

  ],

  "settings":{"theme":"dark","notifications":false}

}');
```

Updating JSON Values (String Manipulation Workaround)

Since Oracle 11g cannot edit JSON with JSON functions, you update by string replace.

Update a field inside JSON

```
UPDATE user_profiles

SET profile_data = REPLACE(profile_data, '"theme":"dark"', '"theme":"light"')

WHERE id = 1;
```

Add a new field (simple string concatenation)

```
UPDATE user_profiles

SET profile_data = SUBSTR(profile_data, 1, LENGTH(profile_data)-1)

                || ',"verified":true}' WHERE id = 2;
```

Replace an array value

```
UPDATE user_profiles

SET profile_data = REPLACE(profile_data, '["SQL","Java","Python"]', '["SQL","Go","Rust"]')
WHERE id = 3;
```

Querying JSON (Text-Based Workarounds) As Oracle 11g cannot extract JSON fields natively, so we use:

- `LIKE`
- `INSTR`
- `REGEXP_LIKE`

Find rows that contain `"active": true`

```
SELECT * FROM user_profiles WHERE profile_data LIKE '%"active":true%';
```

Find users with a specific skill in an array

```
SELECT * FROM user_profiles WHERE profile_data LIKE '%"Python"%';
```

Find rows with a nested key (example: address.city)

```
SELECT * FROM user_profiles WHERE profile_data LIKE '%"city"%';
```

Using regular expressions to match JSON numeric field

```
SELECT * FROM user_profiles WHERE REGEXP_LIKE(profile_data, '"age"\s*:\s*30');
```

Extract a JSON value using REGEXP_SUBSTR (workaround)

Example: Extract "name" from JSON

SELECT id, REGEXP_SUBSTR(profile_data, '"name"\s*:\s*"([^"]+)"', 1, 1, NULL, 1) AS extracted_name FROM user_profiles;

Example output:

```
ID | EXTRACTED_NAME
-------------------
1  | Alice
2  | Bob
3  | Charlie
```

Filtering JSON by Array Length (approximation)

Find rows with multiple skills:

SELECT * FROM user_profiles WHERE REGEXP_LIKE(profile_data, '"skills"\s*:\s*\[.*?,.*?\]');

Detect invalid JSON (simple validation):

Very basic check: JSON should start with { and end with }

SELECT id FROM user_profiles WHERE NOT (profile_data LIKE '{%' AND profile_data LIKE '%}');

SELECT po.po_document.PONumber FROM j_purchaseorder po;

The following query extracts, from each document in JSON column po_document, a scalar value, the JSON number that is the value of field PONumber for the objects in JSON column po_document.

SELECT po.po_document.ShippingInstructions.Phone FROM j_purchaseorder po;

The following query extracts, from each document, an array of JSON phone objects, which is the value of field Phone of the object that is the value of field ShippingInstructions.

SELECT po.po_document.ShippingInstructions.Phone.type FROM j_purchaseorder po;

The following query extracts, from each document, multiple values as an array: the value of field type for each object in array Phone. The returned array is not part of the stored data but is constructed automatically by the query.