Original table:
..The main source table where data is first stored.
Cloned table:
.. A copy created instantly using zero-copy cloning

Assignment 1: Base Table Analysis

1. Load both datasets into Snowflake tables

Employee table:

```
create database emp_table;
create schema emp_schema;

create or replace table employees(
   emp_id int,
   emp_name varchar(100),
   department varchar(100),
   job_title varchar(100),
   salary int,
   location varchar(100),
   hire_date date
);

INSERT INTO employees VALUES
(101,'Ramesh','IT','Data Engineer',95000,'Bangalore','2019-06-15'),
(102,'Anita','HR','HR Executive',65000,'Hyderabad','2020-02-10'),
(103,'John','Finance','Finance Manager',110000,'New York','2016-09-01'),
(104,'Meena','IT','Senior Data Analyst',98000,'Bangalore','2018-11-23'),
(105,'David','IT','Software Engineer',90000,'San Francisco','2021-07-05'),
(106,'Priya','HR','HR Business Partner',72000,'Hyderabad','2019-03-18'),
(107,'Michael','Finance','Account Director',120000,'New York','2015-01-12'),
(108,'Sneha','IT','Junior Developer',78000,'Bangalore','2022-08-09'),
(109,'Arjun','IT','Data Architect',115000,'Bangalore','2017-05-30'),
(110,'Emily','Marketing','Marketing Manager',88000,'Chicago','2018-10-17');

```

Results (just now)

Table | Chart

	status
1	Table EMPLOYEES successfully created.

Department table:

```
CREATE OR REPLACE TABLE departments (
    dept_id INT,
    department varchar(100),
    manager varchar(100),
    budget int,
    location varchar(100)
);

INSERT INTO departments (dept_id, department, manager, budget, location) VALUES
(10, 'IT', 'Arjun Rao', 500000, 'Bangalore'),
(20, 'HR', 'Anita Sharma', 200000, 'Hyderabad'),
(30, 'Finance', 'John Smith', 600000, 'New York'),
(40, 'Marketing', 'Emily Wilson', 300000, 'Chicago');
```

Res (14 hours ago) | Results (just now)

Table | Chart — 1 row — 258ms

	status
1	Table DEPARTMENTS successfully created.

2. Validate row counts and data distribution

Employee table:

```
25
26  SELECT department, COUNT(*) AS emp_count
27  FROM employees
28  GROUP BY department;
29
30
```

Results (just now)

Table | Chart — 4 rows — 409ms

	DEPARTMENT	EMP_COUNT
1	Marketing	1
2	IT	5
3	HR	2
4	Finance	2

Department table:

```
44
45  SELECT department, COUNT(*) AS dept_count
46  FROM departments
47  GROUP BY department;
48
```

Results (just now)

Table | Chart — 4 rows — 137ms

	DEPARTMENT	DEPT_COUNT
1	Marketing	1
2	IT	1
3	HR	1
4	Finance	1

3. Identify candidate tables for cloning

Candidate Tables - source table

1.Employees 2.Departments

4. Explain why these tables are suitable for cloning experiments

- **Small and manageable size**
  Both `employees` and `departments` tables have a limited number of rows, making them ideal for quick cloning and easy observation of changes.

- **Stable base data**
  The data does not change frequently, which is perfect for testing clone behavior without constant data modifications.

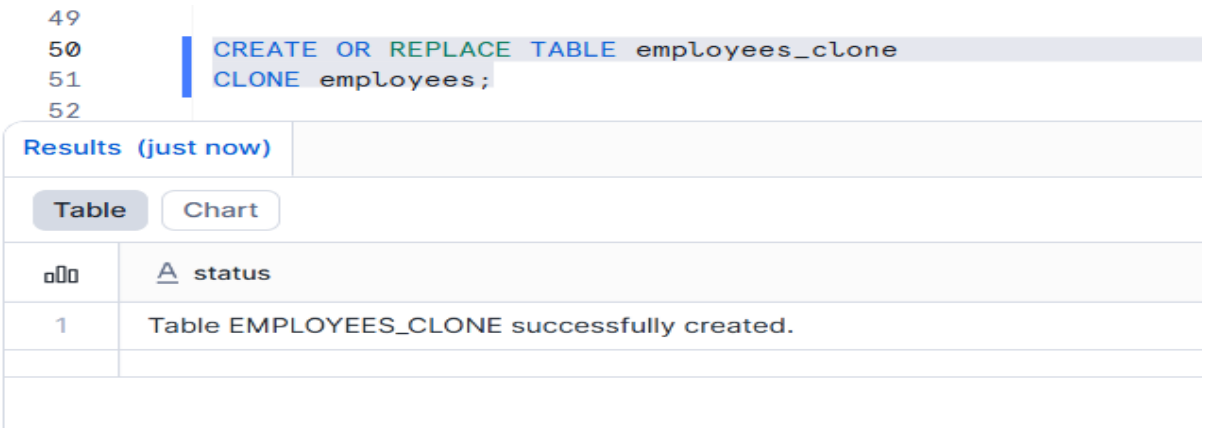
- **Well-structured schema**
  Clearly defined columns (IDs, names, departments, locations) help in easily validating cloned data.

- **No complex transformations**
  These are base tables (not views or CTAS outputs), fully aligned with the rule of using **only cloning**.

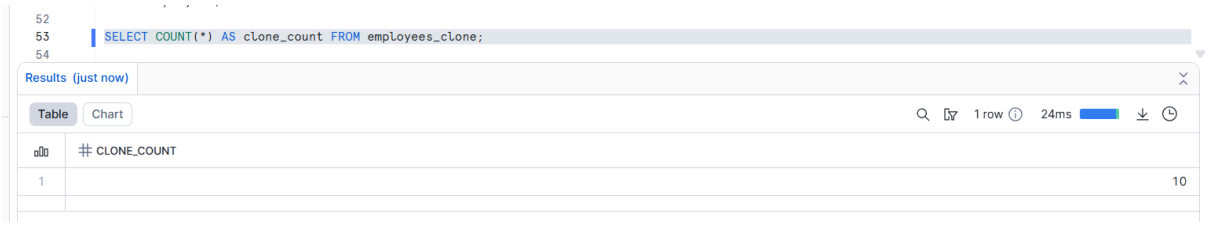
- **Ideal for zero-copy cloning demonstration**
  Snowflake clones share underlying storage initially, allowing students to observe how storage is saved until DML operations occur.

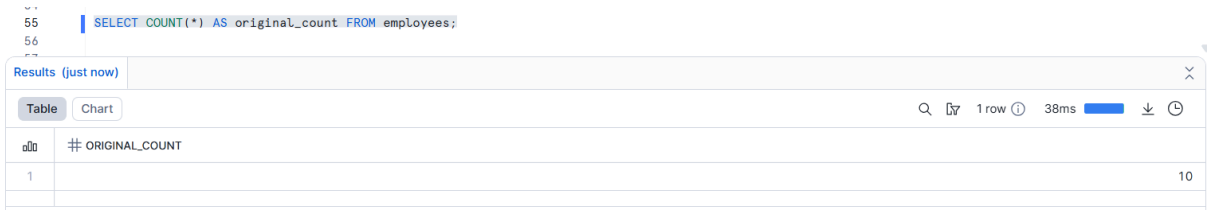
Assignment 2: Table-Level Cloning

1. Clone the employee table



2. Compare:
   - **Row count :** Between clone and original





...Both tables return **10 rows**
…Confirms data consistency between original and clone

- **Query performance:**

```
57  SELECT * FROM employees WHERE department = 'IT';
58
```

Results (just now)

Table | Chart — 5 rows — 33ms

	EMP_ID	EMP_NAME	DEPARTMENT	JOB_TITLE	SALARY	LOCATION	HIRE_DATE
1	101	Ramesh	IT	Data Engineer	95000	Bangalore	2019-06-15
2	104	Meena	IT	Senior Data Analyst	98000	Bangalore	2018-11-23
3	105	David	IT	Software Engineer	90000	San Francisco	2021-07-05
4	108	Sneha	IT	Junior Developer	78000	Bangalore	2022-08-09
5	109	Arjun	IT	Data Architect	115000	Bangalore	2017-05-30

```
59  SELECT * FROM employees_clone WHERE department = 'IT';
60
```

Results (just now)

Table | Chart — 5 rows — 46ms

	EMP_ID	EMP_NAME	DEPARTMENT	JOB_TITLE	SALARY	LOCATION	HIRE_DATE
1	101	Ramesh	IT	Data Engineer	95000	Bangalore	2019-06-15
2	104	Meena	IT	Senior Data Analyst	98000	Bangalore	2018-11-23
3	105	David	IT	Software Engineer	90000	San Francisco	2021-07-05
4	108	Sneha	IT	Junior Developer	78000	Bangalore	2022-08-09
5	109	Arjun	IT	Data Architect	115000	Bangalore	2017-05-30

..Query execution time is nearly identical
..Clone uses the same underlying micro-partitions as the original table

- **Storage usage:**

- BYTES shows how much physical storage the table is using

```
61  SELECT
62      table_name,
63      bytes,
64      row_count
65  FROM information_schema.tables
66  WHERE table_name IN ('EMPLOYEES', 'EMPLOYEES_CLONE');
67
```

Results (just now)

Table | Chart — 2 rows — 1.6s

	TABLE_NAME	BYTES	ROW_COUNT
1	EMPLOYEES_CLONE	3072	10
2	EMPLOYEES	3072	10

3. Explain why clone creation is instantaneous

..Snowflake uses **zero-copy cloning**
..No physical data is copied during clone creation
..Only metadata pointers are created

Assignment 3: Change Isolation Testing

….Checking that changes made in one table do NOT affect another table

1. Update salary data in the original employee table

```
69  UPDATE employees
70  SET salary = salary + 5000
71  WHERE department = 'IT';
72
```

Results (just now)

Table | Chart — 1 row — 215ms

	number of rows updated	number of multi-joined rows updated
1	5	0

```
69  UPDATE employees
70  SET salary = salary + 5000
71  WHERE department = 'IT';
72
73  select*from employees;
```

Results (just now)

Table | Chart — 10 rows — 100ms

	EMP_ID	EMP_NAME	DEPARTMENT	JOB_TITLE	SALARY	LOCATION	HIRE_DATE
	101 – 110	Anita 10.0%, Arjun 10.0%, +8 more	IT 50.0%, Finance 20.0%, +2 more	Account Director 10.0%, Data Architect 10.0%, +8 more	65000 – 125000	Bangalore 40.0%, Hyderabad 20.0%, +3 more	1/12/2015 – 8/9/202
3	106	Priya	HR	HR Business Partner	72000	Hyderabad	2019-03-18
4	107	Michael	Finance	Account Director	120000	New York	2015-01-12
5	101	Ramesh	IT	Data Engineer	105000	Bangalore	2019-06-15
6	104	Meena	IT	Senior Data Analyst	108000	Bangalore	2018-11-23
7	105	David	IT	Software Engineer	100000	San Francisco	2021-07-05
8	108	Sneha	IT	Junior Developer	88000	Bangalore	2022-08-09
9	109	Arjun	IT	Data Architect	125000	Bangalore	2017-05-30
10	110	Emily	Marketing	Marketing Manager	88000	Chicago	2018-10-17

2. Delete records in the cloned table

```
72
73  DELETE FROM employees_clone
74  WHERE department = 'HR';
75
```

Results (just now)

Table | Chart — 1 row — 269ms

	number of rows deleted
1	2

```
73  select*from employees_clone;
74
75  DELETE FROM employees_clone
76  WHERE department = 'HR';
77
```

Results (just now)

Table | Chart — 9 rows — 107ms

	EMP_ID	EMP_NAME	DEPARTMENT	JOB_TITLE	SALARY	LOCATION	HIRE_DATE
1	103	John	Finance	Finance Manager	110000	New York	2016-09-01
2	107	Michael	Finance	Account Director	120000	New York	2015-01-12
3	101	Ramesh	IT	Data Engineer	100000	Bangalore	2019-06-15
4	104	Meena	IT	Senior Data Analyst	103000	Bangalore	2018-11-23
5	105	David	IT	Software Engineer	95000	San Francisco	2021-07-05
6	108	Sneha	IT	Junior Developer	83000	Bangalore	2022-08-09
7	109	Arjun	IT	Data Architect	120000	Bangalore	2017-05-30
8	110	Emily	Marketing	Marketing Manager	88000	Chicago	2018-10-17
9	111	Rahul	IT	Cloud Engineer	105000	Bangalore	2023-01-15

3. Insert new records in the clone

```
INSERT INTO employees_clone
(emp_id, emp_name, department, job_title, salary, location, hire_date)
VALUES
(111, 'Rahul', 'IT', 'Cloud Engineer', 105000, 'Bangalore', '2023-01-15');
```

Results (just now)

Table | Chart — 1 row — 233ms

	number of rows inserted
1	1

```
INSERT INTO employees_clone
(emp_id, emp_name, department, job_title, salary, location, hire_date)
VALUES
(111, 'Rahul', 'IT', 'Cloud Engineer', 105000, 'Bangalore', '2023-01-15');

select*from employees_clone;
```

esults (just now)

Table | Chart — 10 rows — 88ms

	EMP_ID	EMP_NAME	DEPARTMENT	JOB_TITLE	SALARY	LOCATION	HIRE_DATE
	101 – 111	Rahul 20.0%, Arjun 10.0%, +7 more	IT 70.0%, Finance 20.0%, +1 more	Cloud Engineer 20.0%, Account Director 10.0%, +7 more	83000 – 120000	Bangalore 60.0%, New York 20.0%, +2 more	1/12/2015 – 1/15/2023
3	101	Ramesh	IT	Data Engineer	100000	Bangalore	2019-06-15
4	104	Meena	IT	Senior Data Analyst	103000	Bangalore	2018-11-23
5	105	David	IT	Software Engineer	95000	San Francisco	2021-07-05
6	108	Sneha	IT	Junior Developer	83000	Bangalore	2022-08-09
7	109	Arjun	IT	Data Architect	120000	Bangalore	2017-05-30
8	110	Emily	Marketing	Marketing Manager	88000	Chicago	2018-10-17
9	111	Rahul	IT	Cloud Engineer	105000	Bangalore	2023-01-15
10	111	Rahul	IT	Cloud Engineer	105000	Bangalore	2023-01-15

4. Validate isolation between original and clone

Original table:

```
SELECT emp_id, salary
FROM employees
WHERE department = 'IT';
```

esults (just now)

Table | Chart — 5 rows — 42ms

	EMP_ID	SALARY
1	101	110000
2	104	113000
3	105	105000
4	108	93000
5	109	130000

```
SELECT *
FROM employees
WHERE department = 'HR';
```

Results (just now)

Table | Chart — 2 rows — 33ms

	EMP_ID	EMP_NAME	DEPARTMENT	JOB_TITLE	SALARY	LOCATION	HIRE_DATE
1	102	Anita	HR	HR Executive	65000	Hyderabad	2020-02-10
2	106	Priya	HR	HR Business Partner	72000	Hyderabad	2019-03-18

Clone table:

```
SELECT emp_id, salary
FROM employees_clone
WHERE department = 'IT';
```

Results (just now)

Table | Chart — 6 rows — 73ms

	EMP_ID	SALARY
1	101	100000
2	104	103000
3	105	95000
4	108	83000
5	109	120000
6	111	105000

```
select*from employees_clone;

DELETE FROM employees_clone
WHERE department = 'HR';
```

Results (just now)

Table | Chart — 10 rows — 65ms

	EMP_ID (101 – 111)	EMP_NAME (Rahul 20.0%, Arjun 10.0%, +7 more)	DEPARTMENT (IT 70.0%, Finance 20.0%, +1 more)	JOB_TITLE (Cloud Engineer 20.0%, Account Director 10.0%, +7 more)	SALARY (83000 – 120000)	LOCATION (Bangalore 60.0%, New York 20.0%, +2 more)	HIRE_DATE (1/12/2015 – 1/15/2023)
1	103	John	Finance	Finance Manager	110000	New York	2016-09-01
2	107	Michael	Finance	Account Director	120000	New York	2015-01-12
3	101	Ramesh	IT	Data Engineer	100000	Bangalore	2019-06-15
4	104	Meena	IT	Senior Data Analyst	103000	Bangalore	2018-11-23
5	105	David	IT	Software Engineer	95000	San Francisco	2021-07-05
6	108	Sneha	IT	Junior Developer	83000	Bangalore	2022-08-09
7	109	Arjun	IT	Data Architect	120000	Bangalore	2017-05-30
8	110	Emily	Marketing	Marketing Manager	88000	Chicago	2018-10-17

- Update data in the **original table** → clone data remains **unchanged**
- Insert or delete data in the **clone** → original table remains **unchanged**
- Both tables show **different results after changes**

5. Explain backend copy-on-write behavior

Copy-on-write means Snowflake copies data only when changes happen, not when the clone is created

1. Original table and clone **initially share the same data**
2. **No data is copied** at clone creation
3. When a row is **updated, deleted, or inserted**:
   Snowflake **creates a new copy only for the changed data**
4. **Unchanged data remains shared**
5. This ensures **data isolation with minimal storage usage**

Assignment 4: Storage Growth Exploration

*Storage Growth Exploration means checking **when and why storage increases** while working with cloned tables.*

1. Perform DML operations incrementally

Meaning:
Do data changes **one by one**, not all at once.

Example:

- First do **UPDATE**
- Then check storage
- Then do **INSERT**
- Then check storage again

```
101
102  UPDATE employees_clone
103  SET salary = salary + 1000
104  WHERE emp_id = 101;
```

Results (just now)

Table | Chart — 1 row — 265ms

	number of rows updated	number of multi-joined rows updated
1	1	0

```
106
107  INSERT INTO employees_clone
108  VALUES (112, 'Kiran', 'Finance', 'Analyst', 85000, 'Mumbai', '2023-03-01');
109
```

Results (just now)

Table | Chart — 1 row — 242ms

	number of rows inserted
1	1

Final Output:

```
97  SELECT *
98  FROM employees_clone
```

Table | Chart — 11 rows — 96ms

	EMP_ID	EMP_NAME	DEPARTMENT	JOB_TITLE	SALARY	LOCATION	HIRE_DATE
	101 – 112	Rahul 18.2%, Arjun 9.1%, +8 more	IT 63.6%, Finance 27.3%, +1 more	Cloud Engineer 18.2%, Account Director 9.1%, +8 more	83000 – 120000	Bangalore 54.5%, New York 18.2%, +3 more	1/12/2015 – 3/1/2023
3	101	Ramesh	IT	Data Engineer	102000	Bangalore	2019-06-15
4	104	Meena	IT	Senior Data Analyst	103000	Bangalore	2018-11-23
5	105	David	IT	Software Engineer	95000	San Francisco	2021-07-05
6	108	Sneha	IT	Junior Developer	83000	Bangalore	2022-08-09
7	109	Arjun	IT	Data Architect	120000	Bangalore	2017-05-30
8	110	Emily	Marketing	Marketing Manager	88000	Chicago	2018-10-17
9	111	Rahul	IT	Cloud Engineer	105000	Bangalore	2023-01-15
10	111	Rahul	IT	Cloud Engineer	105000	Bangalore	2023-01-15
11	112	Kiran	Finance	Analyst	85000	Mumbai	2023-03-01

2. Track storage before and after each operation

```
110  SELECT table_name, bytes
111  FROM information_schema.tables
112  WHERE table_name IN ('EMPLOYEES', 'EMPLOYEES_CLONE');
113
```

Results (just now)

Table | Chart — 2 rows — 970ms

	TABLE_NAME	BYTES
1	EMPLOYEES	3072
2	EMPLOYEES_CLONE	3072

Before any DML (UPDATE / INSERT / DELETE):
Check storage → **no increase**
After UPDATE:
Check storage again → **storage increases slightly**
After INSERT:
Check storage again → **storage increases more**

3. Identify the exact moment storage increases

Storage increases at the moment when a DML operation is executed (UPDATE, INSERT, or DELETE).

- Creating a clone → **no storage increase**
- Running SELECT queries → **no storage increase**
- First UPDATE / INSERT / DELETE → **storage increases**

4. Explain Snowflake's physical storage strategy

- Snowflake stores data in **small read-only blocks** called **micro-partitions**
- These micro-partitions are **compressed and optimized** automatically
- When a table is cloned, **data is shared**, not copied
- Data blocks are **never changed directly**
- When data is updated, inserted, or deleted:
  ..Snowflake **creates new micro-partitions** for the changed data
  ..Old micro-partitions remain unchanged
- This method is called **copy-on-write**

Assignment 5: Time Travel Cloning

1. Perform multiple updates on the original table

```
114  UPDATE employees
115  SET salary = salary + 3000
116  WHERE department = 'IT';
```

Results (just now)

Table | Chart — 1 row — 2.1s

	number of rows updated	number of multi-joined rows updated
1	5	0

```
117
118  UPDATE employees
119  SET salary = salary + 2000
120  WHERE department = 'HR';
121
```

Results (just now)

Table | Chart — 1 row — 361ms

	number of rows updated	number of multi-joined rows updated
1	2	0

Final Output:

```
114  UPDATE employees
115  SET salary = salary + 3000
116  WHERE department = 'IT';
117
118  UPDATE employees
119  SET salary = salary + 2000
120  WHERE department = 'HR';
121
122  SELECT * FROM employees;
123
```

Results (just now)

Table | Chart — 10 rows — 932ms

	EMP_ID	EMP_NAME	DEPARTMENT	JOB_TITLE	SALARY	LOCATION	HIRE_DATE
	101 – 110	Anita 10.0%, Arjun 10.0%, +8 more	IT 50.0%, Finance 20.0%, +2 more	Account Director 10.0%, Data Architect 10.0%, +8 more	67000 – 133000	Bangalore 40.0%, Hyderabad 20.0%, +3 more	1/12/2015 – 8/9/2022
1	103	John	Finance	Finance Manager	110000	New York	2016-09-01
2	102	Anita	HR	HR Executive	67000	Hyderabad	2020-02-10
3	106	Priya	HR	HR Business Partner	74000	Hyderabad	2019-03-18
4	107	Michael	Finance	Account Director	120000	New York	2015-01-12
5	101	Ramesh	IT	Data Engineer	113000	Bangalore	2019-06-15
6	104	Meena	IT	Senior Data Analyst	116000	Bangalore	2018-11-23
7	105	David	IT	Software Engineer	108000	San Francisco	2021-07-05
8	108	Sneha	IT	Junior Developer	96000	Bangalore	2022-08-09

2. Create clones from:
   - Current state

```
124  CREATE OR REPLACE TABLE employees_clone_current
125  CLONE employees;
126
127
```

Results (just now)

Table | Chart — 1 row — 1.1s

	status
1	Table EMPLOYEES_CLONE_CURRENT successfully created.

- Past state (Time Travel)

```
127  CREATE OR REPLACE TABLE employees_clone_past
128  CLONE employees
129  AT (OFFSET => -300);
130
```

lesults (just now)

Table | Chart — 1 row — 691ms

	status
1	Table EMPLOYEES_CLONE_PAST successfully created.

3. Compare data across versions

```
130
131  SELECT
132      c.emp_id,
133      c.salary AS current_salary,
134      p.salary AS past_salary
135  FROM employees_clone_current c
136  JOIN employees_clone_past p
137  ON c.emp_id = p.emp_id;
```

sults (2 minutes ago)

Table | Chart — 10 rows — 172ms

	EMP_ID	CURRENT_SALARY	PAST_SALARY
1	103	110000	110000
2	102	67000	67000
3	106	74000	74000
4	107	120000	120000
5	101	113000	113000
6	104	116000	116000
7	105	108000	108000
8	108	96000	96000
9	109	133000	133000

4. Explain real-world recovery use cases

- Recover data after **accidental UPDATE or DELETE**
- Restore a table to a **previous correct version**
- Compare data **before and after a mistake**
- Quickly fix **production issues** without reloading data
- Support **audit and investigation** needs

Assignment 6: Schema-Level Cloning

1. Clone a schema containing employee and department tables

```
CREATE OR REPLACE SCHEMA hr_schema_clone
CLONE emp_schema;
```

Table | Chart — 1 row — 2.1s

	status
1	Schema HR_SCHEMA_CLONE successfully created.

2. Modify tables in cloned schema

Employee table:

```
UPDATE hr_schema_clone.employees
SET salary = salary + 3000
WHERE department = 'IT';
```

Table | Chart — 1 row — 257ms

	number of rows updated	number of multi-joined rows updated
1	5	0

```
select * from hr_schema_clone.employees;
```

Results (just now)

Table | Chart — 10 rows — 106ms

	EMP_ID	EMP_NAME	DEPARTMENT	JOB_TITLE	SALARY	LOCATION	HIRE_DATE
2	102	Anita	HR	HR Executive	67000	Hyderabad	2020-02-10
3	106	Priya	HR	HR Business Partner	74000	Hyderabad	2019-03-18
4	107	Michael	Finance	Account Director	120000	New York	2015-01-12
5	101	Ramesh	IT	Data Engineer	119000	Bangalore	2019-06-15
6	104	Meena	IT	Senior Data Analyst	122000	Bangalore	2018-11-23
7	105	David	IT	Software Engineer	114000	San Francisco	2021-07-05
8	108	Sneha	IT	Junior Developer	102000	Bangalore	2022-08-09
9	109	Arjun	IT	Data Architect	139000	Bangalore	2017-05-30
10	110	Emily	Marketing	Marketing Manager	88000	Chicago	2018-10-17

Department table:

```
INSERT INTO hr_schema_clone.departments
VALUES (50, 'Sales', 'Rahul Kumar', 250000, 'Mumbai');
```

Results (just now)

Table | Chart — 1 row — 240ms

	number of rows inserted
1	1

```
155  select * from hr_schema_clone.departments;
```

Results (just now)

Table | Chart — 6 rows — 136ms

	DEPT_ID	DEPARTMENT	MANAGER	BUDGET	LOCATION
1	10	IT	Arjun Rao	500000	Bangalore
2	20	HR	Anita Sharma	200000	Hyderabad
3	30	Finance	John Smith	600000	New York
4	40	Marketing	Emily Wilson	300000	Chicago
5	50	Sales	Rahul Kumar	250000	Mumbai
6	50	Sales	Rahul Kumar	250000	Mumbai

3. Drop tables in original schema

Departments table:

```
156
157  DROP TABLE hr_schema_clone.departments;
158
```

Results (just now)

Table | Chart — 1 row — 68ms

	status
1	DEPARTMENTS successfully dropped.

Employees table:

```
159  DROP TABLE hr_schema_clone.employees;
160
```

Results (just now)

Table | Chart — 1 row — 72ms

	status
1	EMPLOYEES successfully dropped.

4. Observe behavior and explain metadata inheritance

Metadata inheritance means the clone gets its own metadata while sharing data blocks until changes occur.

Observed behavior:

- Changes made in the cloned schema do not affect the original schema
- Dropping tables in the original schema does not delete tables in the clone
- Both schemas work independently

Metadata inheritance (meaning):

- When a schema is cloned, Snowflake copies metadata only, not the actual data
- Both schemas initially point to the same underlying data blocks
- Each schema has its own metadata
- Any change creates new data using copy-on-write

Assignment 7: Database-Level Cloning

1. Clone the entire database

```
161  CREATE OR REPLACE DATABASE emp_db_clone
162  CLONE my_db;
163
```

Results (just now)

Table | Chart — 1 row — 3.3s

	status
1	Database EMP_DB_CLONE successfully created.

2. Test query access and object availability

```
165  USE DATABASE emp_db_clone;
```

Results (just now)

Table | Chart — 1 row — 37ms

	status
1	Statement executed successfully.

```
166  SHOW SCHEMAS;
```

Results (just now)

Table | Chart — 3 rows — 40ms

	created_on	name	is_default	is_current	database_name	owner	comment
1	2025-12-22 07:41:51.176 -080(	INFORMATION_SCHEMA	N	N	EMP_DB_CLONE		Views describing the contents of schemas in th
2	2025-12-22 07:35:48.197 -080(	MY_SCHEMA	N	N	EMP_DB_CLONE	ACCOUNTADMIN	
3	2025-12-22 07:35:48.197 -080(	PUBLIC	N	Y	EMP_DB_CLONE	ACCOUNTADMIN	

```
167  SELECT * FROM hr_schema_clone.employees;
```

Results (just now)

Table | Chart — 10 rows — 765ms

	EMP_ID	EMP_NAME	DEPARTMENT	JOB_TITLE	SALARY	LOCATION	HIRE_DATE
1	103	John	Finance	Finance Manager	110000	New York	2016-09-01
2	102	Anita	HR	HR Executive	67000	Hyderabad	2020-02-10
3	106	Priya	HR	HR Business Partner	74000	Hyderabad	2019-03-18
4	107	Michael	Finance	Account Director	120000	New York	2015-01-12
5	101	Ramesh	IT	Data Engineer	113000	Bangalore	2019-06-15
6	104	Meena	IT	Senior Data Analyst	116000	Bangalore	2018-11-23
7	105	David	IT	Software Engineer	108000	San Francisco	2021-07-05
8	108	Sneha	IT	Junior Developer	96000	Bangalore	2022-08-09
9	109	Arjun	IT	Data Architect	133000	Bangalore	2017-05-30

- All schemas and tables are available
- Data can be queried normally

3. Perform DML on cloned database

```
UPDATE hr_schema_clone.employees
SET salary = salary + 4000
WHERE department = 'IT';
```

esults (just now)

Table | Chart — 1 row — 902ms

	number of rows updated	number of multi-joined rows updated
1	5	0

```
INSERT INTO hr_schema_clone.departments
VALUES (60, 'Support', 'Neha Singh', 180000, 'Pune');
```

Results (just now)

Table | Chart — 1 row — 203ms

	number of rows inserted
1	1

- Changes apply **only to the cloned database**
- Original database remains unchanged

4. Explain how Snowflake isolates environments (Dev/Test/Prod)\

- Snowflake uses **zero-copy database cloning**
- Dev, Test, and Prod databases start by **sharing the same data**
- Each environment has **independent metadata**
- Changes in Dev/Test do **not affect Prod**
- New data is created only when changes happen (**copy-on-write**)

Assignment 9: Performance Comparison

1. Run identical analytical queries on:
   - Original tables

```
SELECT
    department,
    AVG(salary) AS avg_salary
FROM employees
GROUP BY department;
```

Res (13 hours ago) | Results (just now)

Table | Chart — 4 rows — 25ms

	DEPARTMENT	AVG_SALARY
1	IT	95200.000000
2	HR	68500.000000
3	Finance	115000.000000
4	Marketing	88000.000000

   - Cloned tables

```
188  SELECT
189      department,
190      AVG(salary) AS avg_salary
191  FROM employees_clone
192  GROUP BY department;
```

Res (12 hours ago) | Results (just now)

Table | Chart — 4 rows — 27ms

	DEPARTMENT	AVG_SALARY
1	IT	95200.000000
2	HR	68500.000000
3	Finance	115000.000000
4	Marketing	88000.000000

*Both queries return the **same result**.*

2. Compare execution plans

Explain using text - we can identify the query performance.

Original table:

```
194  EXPLAIN USING TEXT
195  SELECT
196      department,
197      AVG(salary)
198  FROM employees
199  GROUP BY department;
```

Res (13 hours ago) | Results (just now)

Table | Chart — 1 row — 32ms

	content
1	GlobalStats: partitionsTotal=1 partitionsAssigned=1 bytesAssigned=3072 Operations: 1:0 →Result EMPLOYEES.DEPARTMENT, SCALED_ROUND_INT_DIVIDE(SUM(EMPLOYEES.SALARY), COU

Clone table:

```
201  EXPLAIN USING TEXT
202  SELECT
203      department,
204      AVG(salary)
205  FROM employees_clone
206  GROUP BY department;
```

Res (13 hours ago) | Results (just now)

Table | Chart — 1 row — 24ms

	content
1	GlobalStats: partitionsTotal=1 partitionsAssigned=1 bytesAssigned=3072 Operations: 1:0 →Result EMPLOYEES_CLONE.DEPARTMENT, SCALED_ROUND_INT_DIVIDE(SUM(EMPLOYEES_CLONE

- Execution plans are **nearly identical**
- Same micro-partitions are scanned
- Same query optimization is applied

3. Explain performance similarities/differences

Performance is almost the same because original and cloned tables share the same data and execution plan; differences appear only after data changes or cache effects.

Assignment 10: Cleanup & Retention

1. Drop original objects

Employees:

```
208  DROP TABLE employees;
```

Res (13 hours ago) | Results (just now)

Table | Chart — 1 row — 98ms

	status
1	EMPLOYEES successfully dropped.

Departments:

```
209  DROP TABLE departments;
210
```

Res (13 hours ago) | Results (just now)

Table | Chart — 1 row — 105ms

	status
1	DEPARTMENTS successfully dropped.

2. Validate cloned object availability

```
211  SELECT * FROM employees_clone;
212
```

Res (13 hours ago) | Results (just now)

Table | Chart — 10 rows — 851ms

	EMP_ID	EMP_NAME	DEPARTMENT	JOB_TITLE	SALARY	LOCATION	HIRE_DATE
1	101	Ramesh	IT	Data Engineer	95000	Bangalore	2019-06-15
2	102	Anita	HR	HR Executive	65000	Hyderabad	2020-02-10
3	103	John	Finance	Finance Manager	110000	New York	2016-09-01
4	104	Meena	IT	Senior Data Analyst	98000	Bangalore	2018-11-23
5	105	David	IT	Software Engineer	90000	San Francisco	2021-07-05
6	106	Priya	HR	HR Business Partner	72000	Hyderabad	2019-03-18
7	107	Michael	Finance	Account Director	120000	New York	2015-01-12

```
213  SELECT * FROM departments_clone;
214
```

Res (13 hours ago) | Results (just now)

Table | Chart — 4 rows — 98ms

	DEPT_ID	DEPARTMENT	MANAGER	BUDGET	LOCATION
1	10	IT	Arjun Rao	500000	Bangalore
2	20	HR	Anita Sharma	200000	Hyderabad
3	30	Finance	John Smith	600000	New York
4	40	Marketing	Emily Wilson	300000	Chicago

3. Drop clone and analyze Time Travel behavior

Drop employees_clone / By using time travel the data is available.After retention period ends, data is permanently removed:

```
215  DROP TABLE employees_clone;
216
217
```

Res (13 hours ago) | Results (just now)

Table | Chart — 1 row — 98ms

	status
1	EMPLOYEES_CLONE successfully dropped.

```
217  SELECT * FROM employees_clone
218  AT (OFFSET => -60);
```

Res (13 hours ago) | Results (just now)

Table | Chart — 10 rows — 81ms

	EMP_ID	EMP_NAME	DEPARTMENT	JOB_TITLE	SALARY	LOCATION	HIRE_DATE
1	101	Ramesh	IT	Data Engineer	95000	Bangalore	2019-06-15
2	102	Anita	HR	HR Executive	65000	Hyderabad	2020-02-10
3	103	John	Finance	Finance Manager	110000	New York	2016-09-01
4	104	Meena	IT	Senior Data Analyst	98000	Bangalore	2018-11-23
5	105	David	IT	Software Engineer	90000	San Francisco	2021-07-05
6	106	Priya	HR	HR Business Partner	72000	Hyderabad	2019-03-18
7	107	Michael	Finance	Account Director	120000	New York	2015-01-12

4. Explain lifecycle and retention rules

- Dropped objects go into **Time Travel**
- Data can be recovered within **retention period**
- After Time Travel → data moves to **Fail-safe**
- After Fail-safe → data is **permanently deleted**
- Retention period depends on **Snowflake edition**