SQL QUERIES

FOR

INCOME RECONCILIATION

FOR

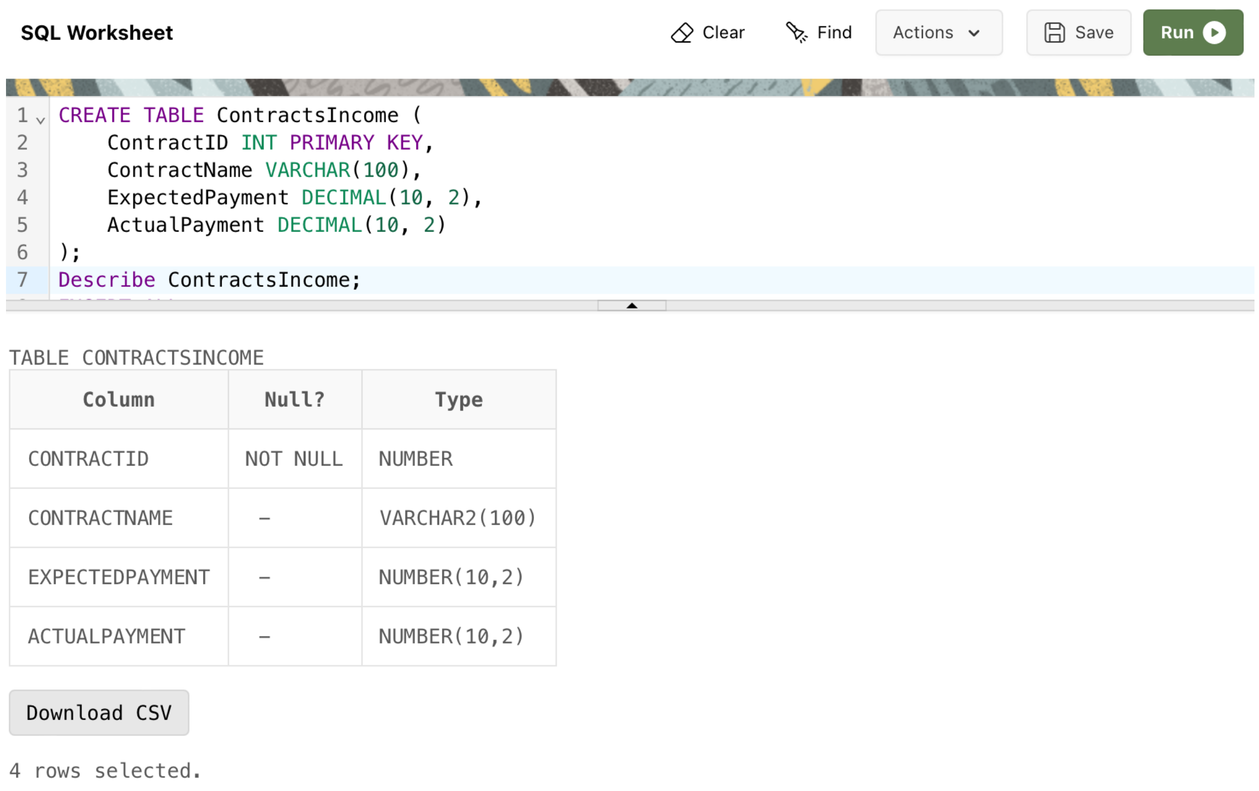
NHS CONTRACTS

- **Objective**: Reconcile actual payments received with the expected income from NHS contracts.

The goal of this project is to streamline the reconciliation process between expected and actual payments from NHS contracts. By identifying discrepancies such as overpayments or underpayments, this project helps ensure that financial records are accurate and that any deviations from expected income are swiftly addressed.

Using SQL queries, the project demonstrates key techniques for detecting discrepancies, classifying payment status, and analyzing patterns in contract payments. This is critical for maintaining financial accountability and providing insights into NHS contract performance. Each query is designed to highlight potential issues and assist in the reconciliation process efficiently.

⇒ <u>QUERY 1</u>



```
CREATE TABLE ContractsIncome (
    ContractID INT PRIMARY KEY,
    ContractName VARCHAR(100),
    ExpectedPayment DECIMAL(10, 2),
    ActualPayment DECIMAL(10, 2)
);
Describe ContractsIncome;
```

TABLE CONTRACTSINCOME

Column	Null?	Type
CONTRACTID	NOT NULL	NUMBER
CONTRACTNAME	–	VARCHAR2(100)
EXPECTEDPAYMENT	–	NUMBER(10,2)
ACTUALPAYMENT	–	NUMBER(10,2)

```
4 rows selected.
```

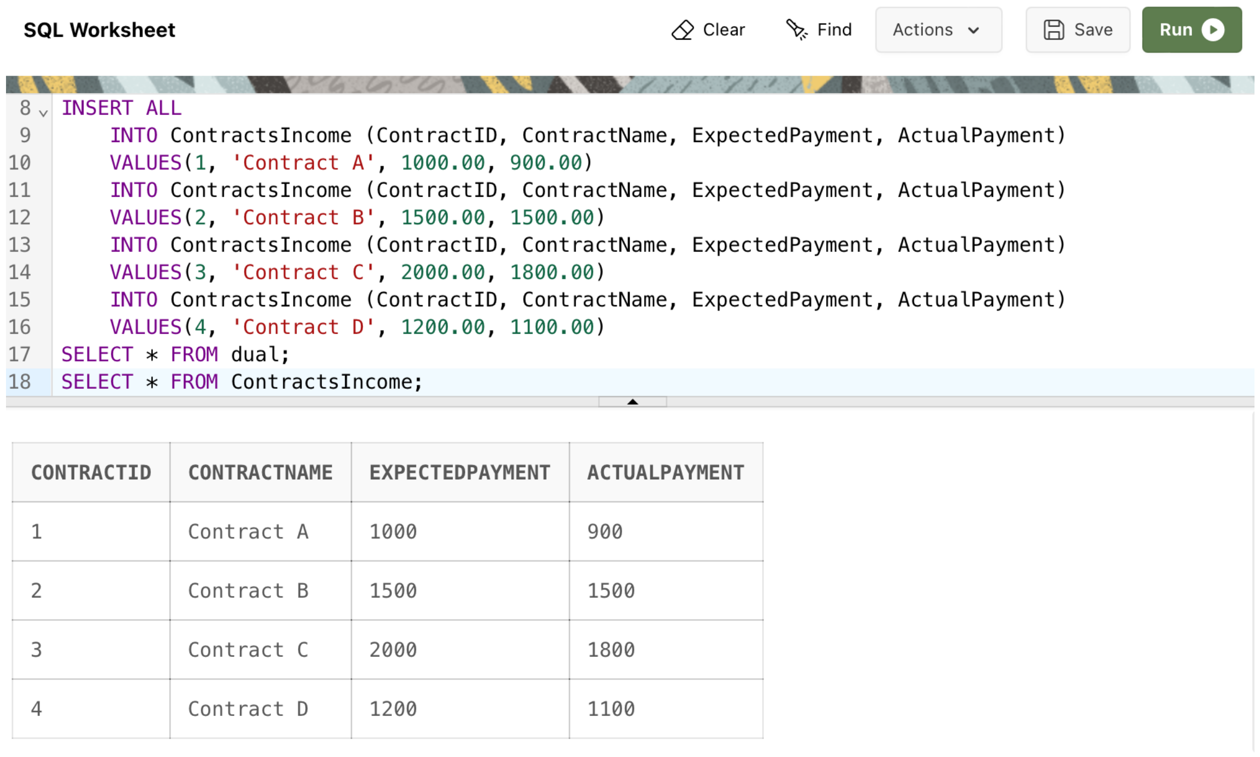
```
CREATE TABLE ContractsIncome (
    ContractID INT PRIMARY KEY,
    ContractName VARCHAR(100),
    ExpectedPayment DECIMAL(10, 2),
    ActualPayment DECIMAL(10, 2)
)
```

Description:
This SQL statement creates the `ContractsIncome` table, which stores data related to NHS contracts. The table includes the following columns:

- `ContractID`: A unique identifier for each contract, defined as the primary key.
- `ContractName`: The name of the contract, stored as a string with a maximum length of 100 characters.
- `ExpectedPayment`: The payment expected from the contract, stored as a decimal value with up to 10 digits and 2 decimal places.
- `ActualPayment`: The payment actually received from the contract, stored similarly as a decimal value.

This table structure forms the foundation for analyzing and reconciling payments for each NHS contract.

⇒ QUERY 2



```
INSERT ALL
    INTO ContractsIncome (ContractID, ContractName, ExpectedPayment, ActualPayment)
    VALUES(1, 'Contract A', 1000.00, 900.00)
    INTO ContractsIncome (ContractID, ContractName, ExpectedPayment, ActualPayment)
    VALUES(2, 'Contract B', 1500.00, 1500.00)
    INTO ContractsIncome (ContractID, ContractName, ExpectedPayment, ActualPayment)
    VALUES(3, 'Contract C', 2000.00, 1800.00)
    INTO ContractsIncome (ContractID, ContractName, ExpectedPayment, ActualPayment)
    VALUES(4, 'Contract D', 1200.00, 1100.00)
SELECT * FROM dual;
SELECT * FROM ContractsIncome;
```

CONTRACTID	CONTRACTNAME	EXPECTEDPAYMENT	ACTUALPAYMENT
1	Contract A	1000	900
2	Contract B	1500	1500
3	Contract C	2000	1800
4	Contract D	1200	1100

```
INSERT ALL
  INTO ContractsIncome (ContractID, ContractName, ExpectedPayment, ActualPayment)
  VALUES(1, 'Contract A', 1000.00, 900.00)
  INTO ContractsIncome (ContractID, ContractName, ExpectedPayment, ActualPayment)
  VALUES(2, 'Contract B', 1500.00, 1500.00)
  INTO ContractsIncome (ContractID, ContractName, ExpectedPayment, ActualPayment)
  VALUES(3, 'Contract C', 2000.00, 1800.00)
  INTO ContractsIncome (ContractID, ContractName, ExpectedPayment, ActualPayment)
  VALUES(4, 'Contract D', 1200.00, 1100.00)
SELECT * FROM dual;
```

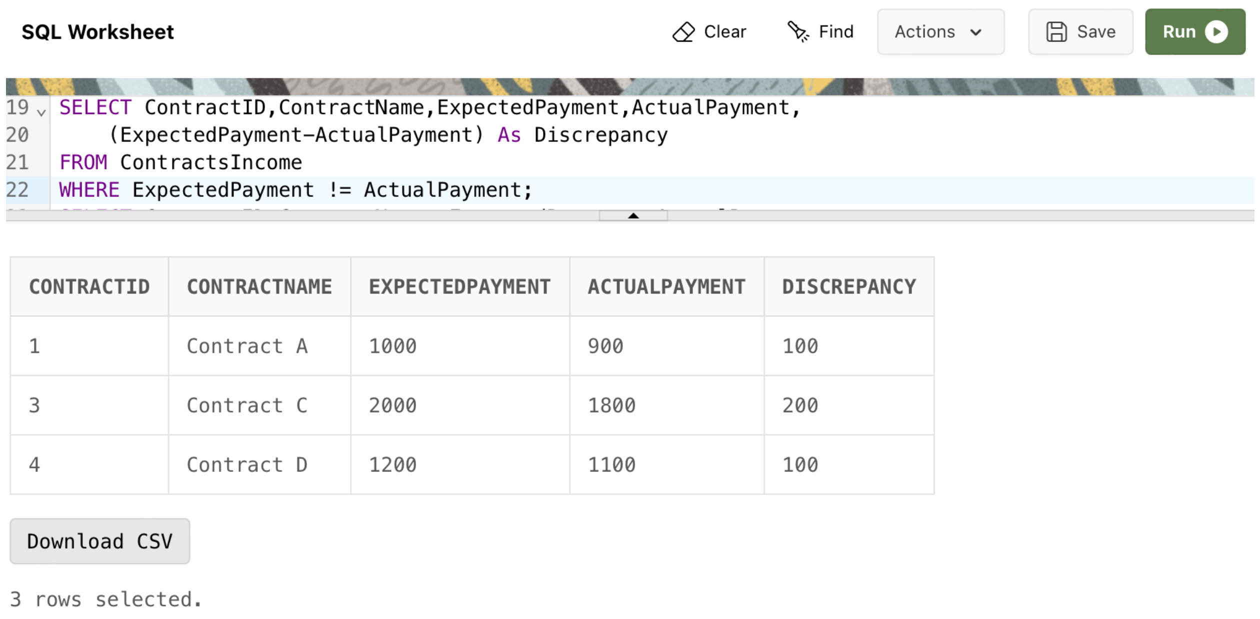
Description:
This SQL statement inserts multiple records into the `ContractsIncome` table in a single operation using the `INSERT ALL` syntax. It adds four contracts, each with specific details regarding their ID, name, expected payment, and actual payment. The inserted records include:

- **Contract A** with an expected payment of £1,000.00 and an actual payment of £900.00 (indicating an underpayment).
- **Contract B** with an expected payment of £1,500.00 and an actual payment of £1,500.00 (indicating no discrepancy).
- **Contract C** with an expected payment of £2,000.00 and an actual payment of £1,800.00 (indicating an underpayment).
- **Contract D** with an expected payment of £1,200.00 and an actual payment of £1,100.00 (indicating an underpayment).

The use of `SELECT * FROM dual` is a common practice in Oracle databases to complete the `INSERT ALL` statement. This operation initializes the table with example data for further analysis and reconciliation.

⇒ **QUERY 3**

Identifying Payment Discrepancies



```sql
SELECT ContractID,ContractName,ExpectedPayment,ActualPayment,
    (ExpectedPayment-ActualPayment) As Discrepancy
FROM ContractsIncome
WHERE ExpectedPayment != ActualPayment;
```

CONTRACTID	CONTRACTNAME	EXPECTEDPAYMENT	ACTUALPAYMENT	DISCREPANCY
1	Contract A	1000	900	100
3	Contract C	2000	1800	200
4	Contract D	1200	1100	100

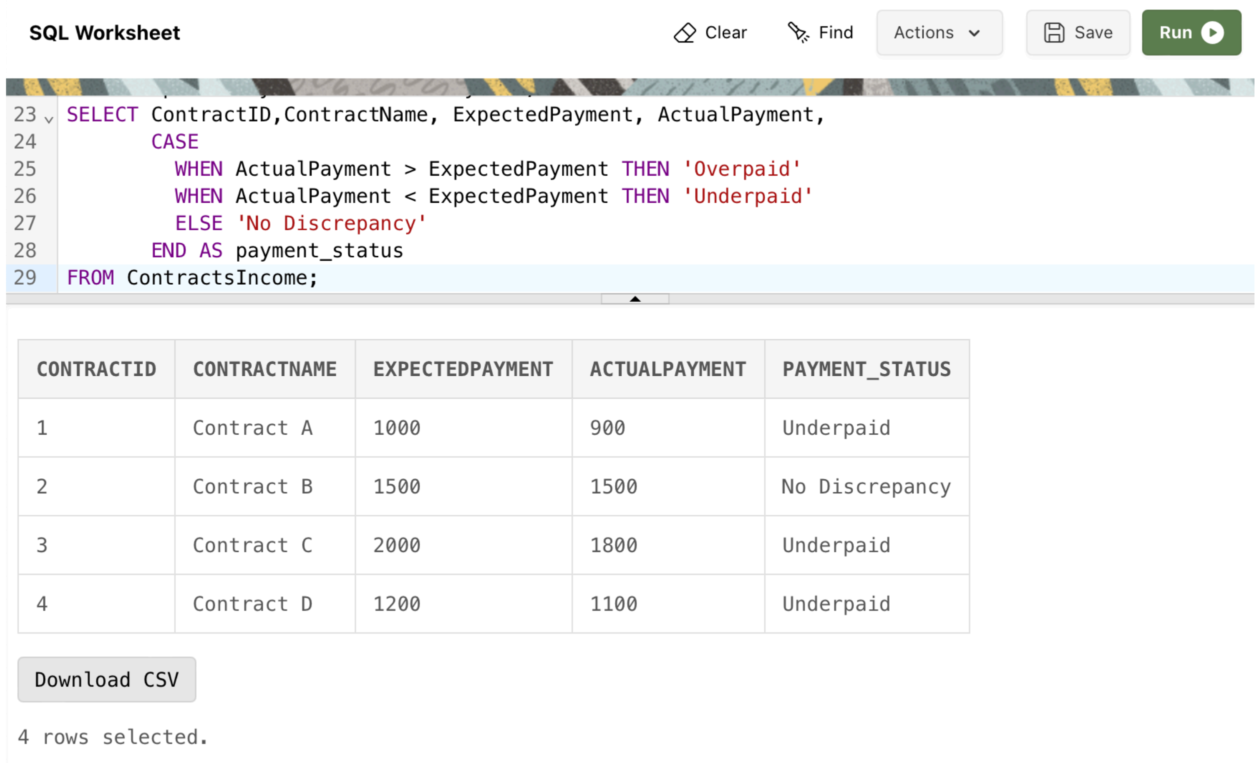
```
3 rows selected.
```

```sql
Select ContractID,ContractName,ExpectedPayment,ActualPayment,
   (ExpectedPayment-ActualPayment) As Discrepancy
From ContractsIncome
Where ExpectedPayment != ActualPayment;
```

Description: This query retrieves the contract ID, contract name, expected payment, actual payment, and the discrepancy between the two. It filters the results to only show contracts where the expected and actual payments are not equal, identifying discrepancies that need attention.

⇒ <u>**QUERY 4**</u>

Classifying Overpayments and Underpayments



```
SELECT ContractID,ContractName, ExpectedPayment, ActualPayment,
       CASE
         WHEN ActualPayment > ExpectedPayment THEN 'Overpaid'
         WHEN ActualPayment < ExpectedPayment THEN 'Underpaid'
         ELSE 'No Discrepancy'
       END AS payment_status
FROM ContractsIncome;
```

CONTRACTID	CONTRACTNAME	EXPECTEDPAYMENT	ACTUALPAYMENT	PAYMENT_STATUS
1	Contract A	1000	900	Underpaid
2	Contract B	1500	1500	No Discrepancy
3	Contract C	2000	1800	Underpaid
4	Contract D	1200	1100	Underpaid

```
4 rows selected.
```

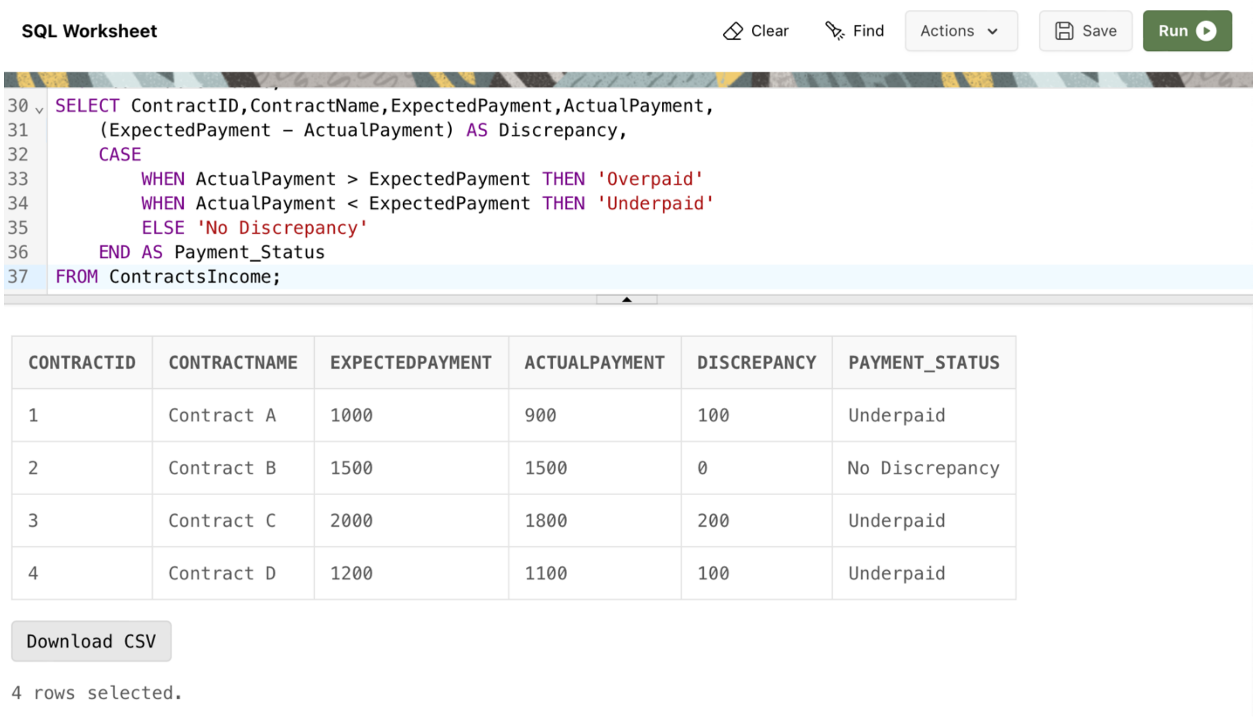
```
SELECT ContractID,ContractName, ExpectedPayment, ActualPayment,
      CASE
        WHEN ActualPayment > ExpectedPayment THEN 'Overpaid'
        WHEN ActualPayment < ExpectedPayment THEN 'Underpaid'
        ELSE 'No Discrepancy'
      END AS payment_status
FROM ContractsIncome;
```

Description: This query includes a `CASE` statement to classify contracts based on their payment status. It categorizes contracts as "Overpaid" if the actual payment exceeds the expected amount, "Underpaid" if it's less, and "No Discrepancy" if they match. This provides insight into the type of discrepancy for each contract.

⇒ QUERY 5

Generating Payment Status Report

This query generates a detailed report that includes payment status and discrepancies.



```
SELECT ContractID,ContractName,ExpectedPayment,ActualPayment,
    (ExpectedPayment - ActualPayment) AS Discrepancy,
    CASE
        WHEN ActualPayment > ExpectedPayment THEN 'Overpaid'
        WHEN ActualPayment < ExpectedPayment THEN 'Underpaid'
        ELSE 'No Discrepancy'
    END AS Payment_Status
FROM ContractsIncome;
```

CONTRACTID	CONTRACTNAME	EXPECTEDPAYMENT	ACTUALPAYMENT	DISCREPANCY	PAYMENT_STATUS
1	Contract A	1000	900	100	Underpaid
2	Contract B	1500	1500	0	No Discrepancy
3	Contract C	2000	1800	200	Underpaid
4	Contract D	1200	1100	100	Underpaid

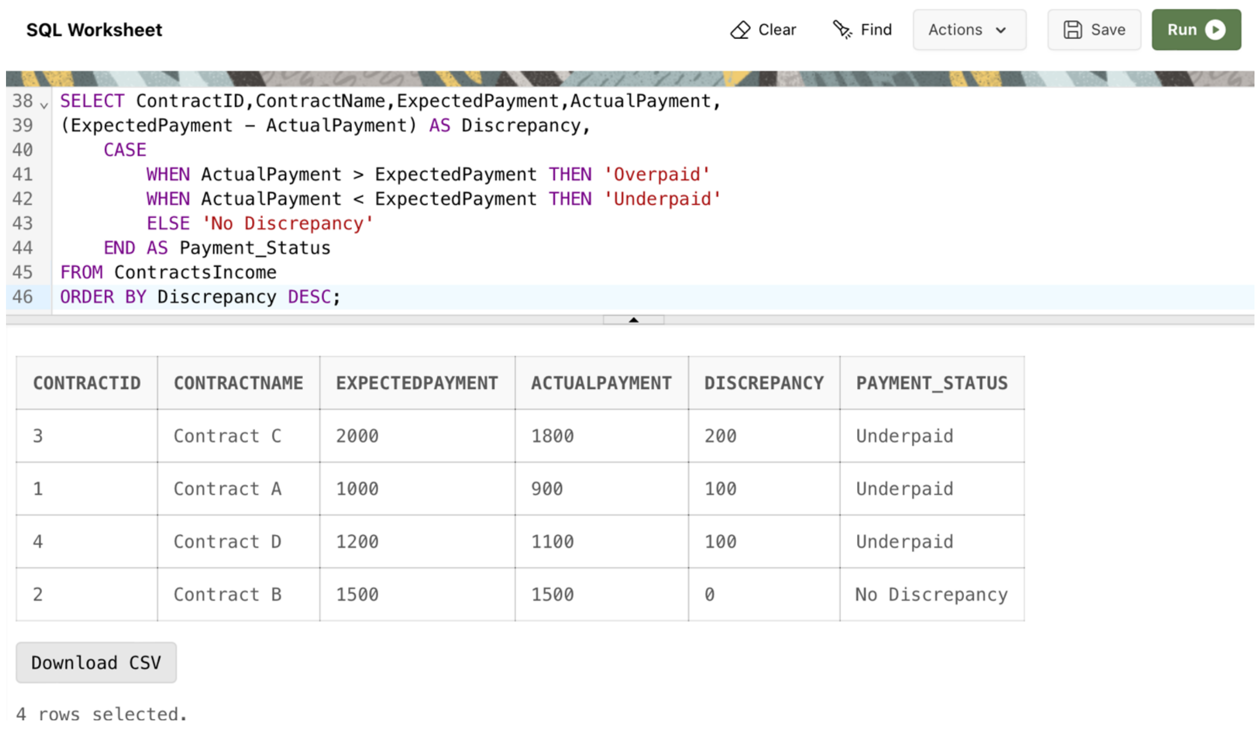
```
4 rows selected.
```

```
SELECT ContractID,ContractName,ExpectedPayment,ActualPayment,
   (ExpectedPayment - ActualPayment) AS Discrepancy,
   CASE
      WHEN ActualPayment > ExpectedPayment THEN 'Overpaid'
      WHEN ActualPayment < ExpectedPayment THEN 'Underpaid'
      ELSE 'No Discrepancy'
   END AS Payment_Status
FROM ContractsIncome;
```

Description:
This query retrieves detailed payment information for each contract, including contract ID, contract name, expected payment, actual payment, and the discrepancy between the two amounts. It also uses a `CASE` statement to categorize each contract into "Overpaid," "Underpaid," or "No Discrepancy" based on the comparison between actual and expected payments. This provides a comprehensive view of each contract's payment status and highlights potential issues for reconciliation.

⇒ **QUERY 6**

Sorting Discrepancies by Amount



```
SELECT ContractID,ContractName,ExpectedPayment,ActualPayment,
(ExpectedPayment - ActualPayment) AS Discrepancy,
    CASE
        WHEN ActualPayment > ExpectedPayment THEN 'Overpaid'
        WHEN ActualPayment < ExpectedPayment THEN 'Underpaid'
        ELSE 'No Discrepancy'
    END AS Payment_Status
FROM ContractsIncome
ORDER BY Discrepancy DESC;
```

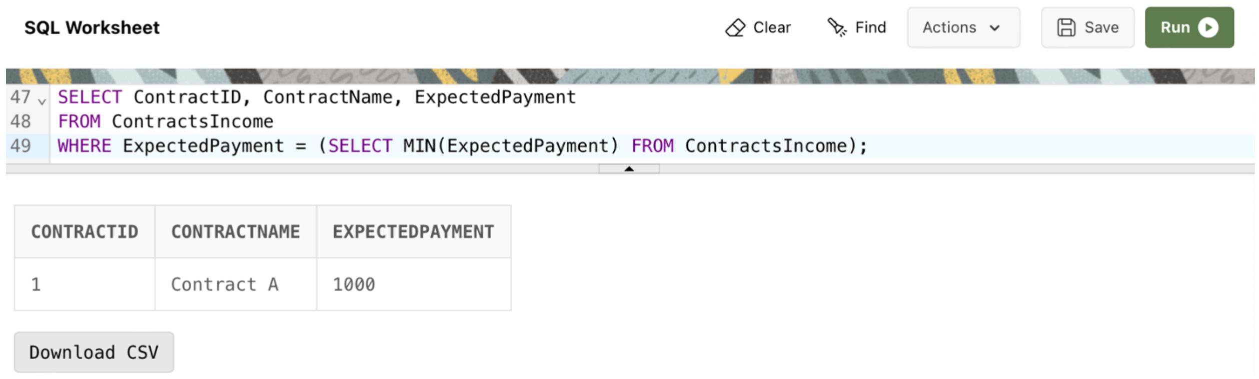
CONTRACTID	CONTRACTNAME	EXPECTEDPAYMENT	ACTUALPAYMENT	DISCREPANCY	PAYMENT_STATUS
3	Contract C	2000	1800	200	Underpaid
1	Contract A	1000	900	100	Underpaid
4	Contract D	1200	1100	100	Underpaid
2	Contract B	1500	1500	0	No Discrepancy

```
4 rows selected.
```

```
SELECT ContractID,ContractName,ExpectedPayment,ActualPayment,
(ExpectedPayment - ActualPayment) AS Discrepancy,
    CASE
        WHEN ActualPayment > ExpectedPayment THEN 'Overpaid'
        WHEN ActualPayment < ExpectedPayment THEN 'Underpaid'
        ELSE 'No Discrepancy'
    END AS Payment_Status
FROM ContractsIncome
ORDER BY Discrepancy DESC;
```

Description: This query combines discrepancy calculation and payment classification, then sorts the results in descending order of discrepancy. This highlights the contracts with the largest discrepancies, allowing for prioritized investigation.

⇒ QUERY 7



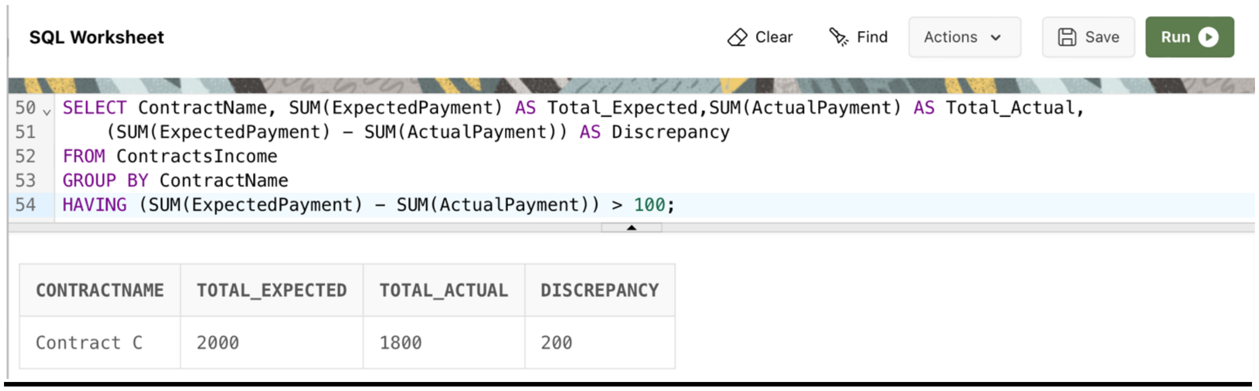
```
SELECT ContractID, ContractName, ExpectedPayment
FROM ContractsIncome
WHERE ExpectedPayment = (SELECT MIN(ExpectedPayment) FROM ContractsIncome);
```

CONTRACTID	CONTRACTNAME	EXPECTEDPAYMENT
1	Contract A	1000

SELECT ContractID, ContractName, ExpectedPayment
FROM ContractsIncome
WHERE ExpectedPayment = (SELECT MIN(ExpectedPayment) FROM ContractsIncome);

Description: This query identifies the contract with the smallest expected payment by comparing each contract's expected payment to the minimum expected payment in the dataset. It helps find outliers that may indicate unusually low contracted amounts.

⇒ QUERY 8



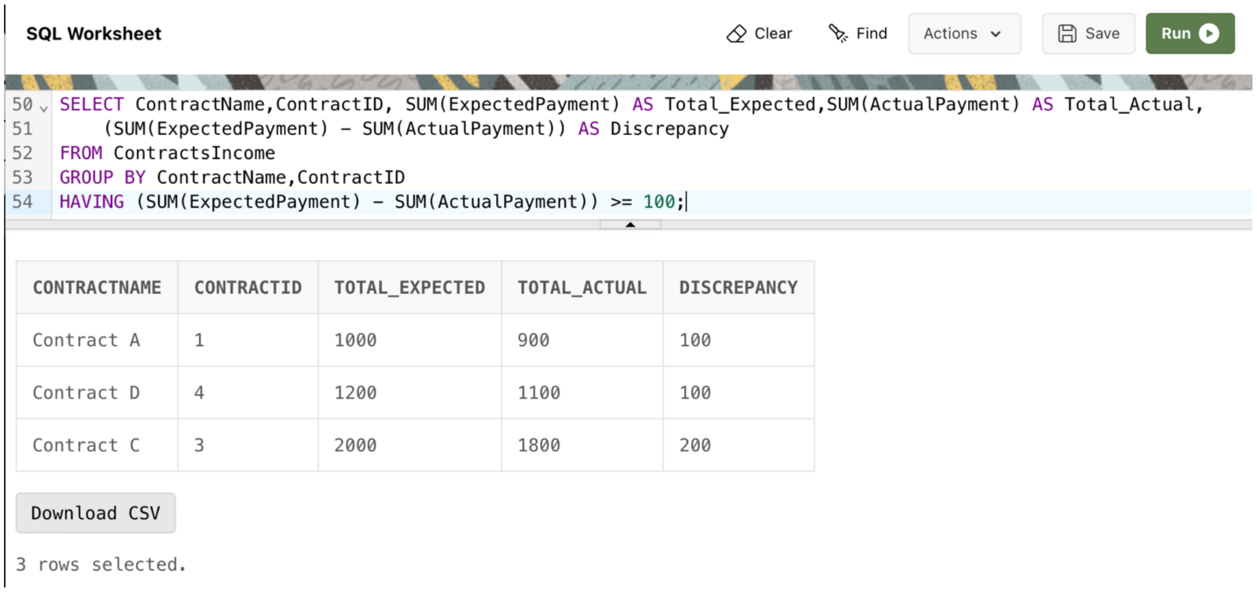
```
SELECT ContractName, SUM(ExpectedPayment) AS Total_Expected,SUM(ActualPayment) AS Total_Actual,
    (SUM(ExpectedPayment) - SUM(ActualPayment)) AS Discrepancy
FROM ContractsIncome
GROUP BY ContractName
HAVING (SUM(ExpectedPayment) - SUM(ActualPayment)) > 100;
```

CONTRACTNAME	TOTAL_EXPECTED	TOTAL_ACTUAL	DISCREPANCY
Contract C	2000	1800	200

SELECT ContractName, SUM(ExpectedPayment) AS Total_Expected, SUM(ActualPayment) AS Total_Actual, (SUM(ExpectedPayment) - SUM(ActualPayment)) AS Discrepancy
FROM ContractsIncome
GROUP BY ContractName
HAVING (SUM(ExpectedPayment) - SUM(ActualPayment)) > 100;

Description:
This query calculates the total expected and actual payments for each contract and identifies discrepancies where the difference between the sums exceeds 100. It helps to highlight contracts with significant payment discrepancies, focusing on those that may require further investigation or reconciliation. The GROUP BY clause ensures that the data is aggregated at the contract level, and the HAVING clause filters out contracts with minor discrepancies.

⇒ QUERY 9



```
SELECT ContractName,ContractID, SUM(ExpectedPayment) AS Total_Expected,SUM(ActualPayment) AS Total_Actual,
    (SUM(ExpectedPayment) - SUM(ActualPayment)) AS Discrepancy
FROM ContractsIncome
GROUP BY ContractName,ContractID
HAVING (SUM(ExpectedPayment) - SUM(ActualPayment)) >= 100;
```

CONTRACTNAME	CONTRACTID	TOTAL_EXPECTED	TOTAL_ACTUAL	DISCREPANCY
Contract A	1	1000	900	100
Contract D	4	1200	1100	100
Contract C	3	2000	1800	200

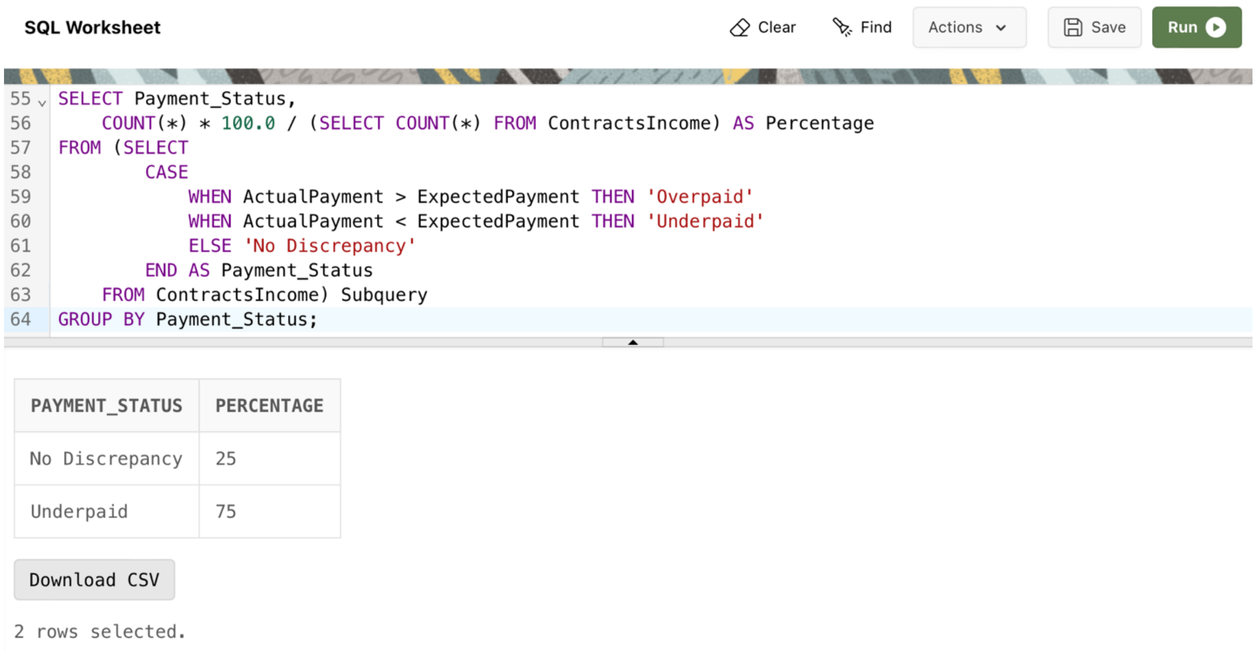
Download CSV

```
3 rows selected.
```

SELECT ContractName,ContractID, SUM(ExpectedPayment) AS Total_Expected,SUM(ActualPayment) AS Total_Actual,
(SUM(ExpectedPayment) - SUM(ActualPayment)) AS Discrepancy
FROM ContractsIncome
GROUP BY ContractName,ContractID
HAVING (SUM(ExpectedPayment) - SUM(ActualPayment)) >= 100

Description:
This query provides a detailed summary of payment discrepancies for each contract by including both the contract name and contract ID. It calculates the total expected and actual payments for each contract and identifies discrepancies of 100 or more. The `GROUP BY` clause ensures the results are grouped by both contract name and ID, allowing for a more granular view of the data. The `HAVING` clause filters the results to show only contracts with significant discrepancies, making it easier to focus on high-priority issues.

⇒ QUERY 10



```sql
SELECT Payment_Status,
    COUNT(*) * 100.0 / (SELECT COUNT(*) FROM ContractsIncome) AS Percentage
FROM (SELECT
        CASE
            WHEN ActualPayment > ExpectedPayment THEN 'Overpaid'
            WHEN ActualPayment < ExpectedPayment THEN 'Underpaid'
            ELSE 'No Discrepancy'
        END AS Payment_Status
    FROM ContractsIncome) Subquery
GROUP BY Payment_Status;
```

PAYMENT_STATUS	PERCENTAGE
No Discrepancy	25
Underpaid	75

Download CSV

2 rows selected.

```
SELECT Payment_Status,
    COUNT(*) * 100.0 / (SELECT COUNT(*) FROM ContractsIncome) AS
Percentage
FROM (SELECT
        CASE
            WHEN ActualPayment > ExpectedPayment THEN 'Overpaid'
            WHEN ActualPayment < ExpectedPayment THEN 'Underpaid'
            ELSE 'No Discrepancy'
        END AS Payment_Status
    FROM ContractsIncome) Subquery
GROUP BY Payment_Status;
```

Description:
This query calculates the percentage of contracts that fall under different payment statuses: "Overpaid," "Underpaid," or "No Discrepancy." It uses a subquery to classify each contract based on the relationship between actual and expected payments, and then groups the results by `Payment_Status`. The percentage for each status is computed by dividing the count of each status by the total number of contracts, giving insight into the distribution of discrepancies in the dataset.