Financial Data Analysis for IDA

Karthika Rajan Nair

karthikanair311@gmail.com

***Abstract*—This project establishes a database for analyzing IDA credits and grants, targeting financial analysts, project managers, and policymakers in development sectors. Focusing on credit status, financial metrics, and project timelines, it provides insights for optimizing resource allocation and project impact.**

I. Introduction

International Development Association (IDA) credits and grants play a crucial role in supporting economic and social development in the world's poorest countries. By providing financial assistance with highly favorable terms, the IDA enables recipient countries to undertake projects that can lead to significant improvements in infrastructure, education, healthcare, and overall economic well-being. This proposal outlines a project aimed at analyzing the distribution and impact of IDA credits and grants across various regions and countries, focusing on credit status and financial metrics.

II. Problem Statement

IDA allocates credits and grants across regions around the world and ensuring that these allocations are effectively utilized for development is crucial. Understanding the distribution, analyzing the impact and evaluating efficiency is challenging. Through this project, we are aiming to conduct a thorough analysis of this data focusing on the following:

- The distribution of credits and grants across regions and countries.
- The status of credits (e.g., fully repaid, repaying, etc.) and their implications on financial health.
- Financial metrics, including original principal amounts, disbursed amounts, repaid amounts, and due amounts.

A. Reasons to opt Database instead of Excel

- Volume of Data: Can handle large volumes of data (millions to billions of rows) efficiently without performance degradation.
- Concurrent Access: Supports multiple users accessing and modifying the data concurrently, ensuring data integrity and consistency.
- Data Integrity and Validation: Offers robust data integrity features, including constraints (e.g., primary keys, foreign keys, unique constraints) and transactions, ensuring accurate and consistent data.
- Scalability: Easily scalable to accommodate growing data needs, both in terms of storage and performance.
- Security: Provides advanced security features, including user authentication, encryption, and fine-grained access controls to secure sensitive data.
- Complex Queries and Analysis: Supports complex queries, aggregations, and transactions, enabling sophisticated data manipulation, analysis, and reporting.
- Automation and Integration: Easily integrates with other applications and services, supporting automation through scripting, APIs, and stored procedures.
- Data Recovery and Backup: Supports robust backup and recovery mechanisms, allowing for data restoration in case of loss or corruption.

III. Target User

A. Database Users

The users of the database will be financial analysts and project managers, the people who are working with the International Development Association (IDA).

- Financial Analysts: These people will analyse the financial metrics like the credit and grants of IDA, which includes disbursed amounts, repaid amounts and the due amounts. They can use this to get the patterns in the IDA funded projects, identify the patterns in the repayment structure and make decisions accordingly.
- Project Managers: They can use this database to track and manage the status of IDA funded projects. It will help them to know the details of Project performance, Credit transaction status. It will help in evaluation and planning of the projects.

B. Database Administrator

The database will be administered by a team of database administrators of the international development organization.

- Database Maintenance: Frequent updates and maintaining the database will guarantee accuracy like adding and updating the necessary data.
- User Support: Gives support to the clients by letting them know how to use the database, and how to work with the queries.

C. Real-Life Scenario

Consider a scenario where an international development organization is overseeing multiple IDA-funded projects across the countries in Africa. Financial analysts in the organization can use our database to perform in-depth analysis of financial

status and monetary transaction details pertaining to specific region-country combination. The can identify any projects that are behind on repayments and assess the overall financial health.

Project managers can use the database to oversee the allocation of funds in alignment with project milestones.

The Database Administrators can ensure that the database stays up-to-date, is secure, and maintains high performance, facilitating the smooth operation of the organization's development programs.

IV. Functional Dependencies and BCNF violations

As a part of this project, we aim to ensure the normalization of the database schema to Boyce-Codd Normal Form (BCNF). Below, are the details of the transformation process from the initial schema to the final schema where all relations are in BCNF.

A. Countries

- Country_ID →Region, Country_Code, Country
- Country_Code →Region, Country, Country_ID

BCNF Verification: The primary key is Country_ID. Country_Code appear to be a candidate key as well since it uniquely identifies all attributes. So, the table is in BCNF.

B. Credit_Status_Details

- Country_Status_ID →Credit_Status

BCNF Verification: This is a two attribute relation and will surely be in BCNF.

C. Project_Details

- Project_Detail_ID →Project_ID, Project_Name, Country_ID, Credit_Status_ID

BCNF Verification: Here Project_Detail_ID is the primary key which uniquely identifies all the functional dependencies in the table. So, this table is already in BCNF

D. Credit_Transactions

- Financial_ID →Project_Detail_ID, Credit_Status_ID, Original_Principal_Amount, Disbursed_Amount, Repaid_to_IDA

BCNF Verification: The primary key is Financial_ID, which is the determinant for all other attributes. This table is in BCNF.

E. Borrowers

- Borrower_ID →Country_ID, Borrower_Obligation

BCNF Verification: The primary key is Borrower_ID. There are no other functional dependencies and so this table is in BCNF.

V. ER Diagram

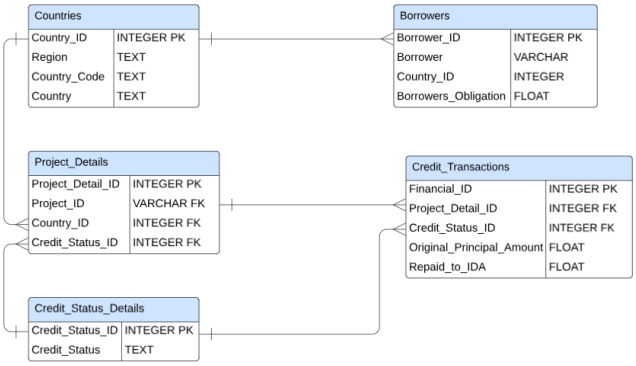


Fig. 1. ER Diagram for Financial Data Analysis

VI. List of Relations and their Attributes

A. Countries Table

- Primary Key: 'Country_ID' - Unique identifier for each country, ensuring each record represents a distinct country.
- Foreign Key: None

B. Credit_Status_Details Table

- Primary Key: 'Credit_Status_ID' - Unique identifier for each credit status, ensuring each record represents a distinct status.
- Foreign Key: None

C. Project_Details Table

- Primary Key: 'Project_Detail_ID' - Unique identifier for each project, ensuring each record represents a distinct project.
- Foreign Key:
  - 'Project_ID' - References the Project_Info table to establish a relationship which enables to join two tables to get the Project_Name for each detail record.
  - 'Project_Name' - Short descriptive project name.
  - 'Country_ID' - References the Countries table to establish a relationship between each project and the country it is associated with.
  - 'Credit_Status_ID' - References the Credit_Status_Details table to establish a connection between the status of the credit for a particular project.

D. Credit_Transactions Table

- Primary Key: 'Financial_ID' - Unique identifier for each financial transaction, ensuring each record represents a distinct transaction.
- Foreign Key:
  - 'Project_Detail_ID' - References the Projects table to link each financial transaction to a specific project.
  - 'Credit_Status_ID' - References the Credit_Status_Details table to establish a connection between the status of the credit.

E. Borrowers Table

- Primary Key: 'Borrower_ID' - Unique identifier for each borrower, ensuring each record represents a distinct borrower entity.
- Foreign Key:

– 'Country_ID' - References the Countries table to link each borrower to a specific country.

VII. Description of Each Attribute

A. Countries Table

- 'Country_ID' (INTEGER PRIMARY KEY) - Unique identifier for each country, ensuring each record represents a distinct country.
- 'Region' (TEXT NOT NULL) - Country lending is grouped into regions based on the current World Bank administrative (rather than geographic) region where project implementation takes place. The Other Region is used for loans to the IFC.
- 'Country_Code' (TEXT NOT NULL) - Country Code according to the World Bank country list. Might be different from the ISO country code.
- 'Country' (TEXT NOT NULL) - Country to which loan has been issued. Loans to the IFC are included under the country "World".

B. Credit_Status_Details Table

- 'Credit_Status_ID' (INTEGER PRIMARY KEY) - Unique identifier for each credit status, ensuring each record represents a distinct status.
- 'Credit_Status' (TEXT NOT NULL) - Status of the loan.See Data Dictionary attached in the About section or Data Dictionary dataset available from the list of all datasets for status descriptions.

C. Project_Details Table

- 'Project_Detail_ID' (INTEGER PRIMARY KEY) - Unique identifier for each project, ensuring each record represents a distinct project.
- 'Project_ID' (VARCHAR) - A Bank project is referenced by a project ID (Pxxxxxxx). More than one loan, credit, or grant may be associated with one Project ID.
- 'Project_Name' (TEXT NOT NULL) - Short descriptive project name.
- 'Country_ID' (INTEGER FOREIGN KEY) - References the Countries table to establish a relationship between each project and the country it is associated with.
- 'Credit_Status_ID' (INTEGER FOREIGN KEY) - References the Credit_Status_Details table to establish a connection between the status of the credit.

D. Credit_Transactions Table

- 'Financial_ID' (INTEGER NOT NULL PRIMARY KEY) - Unique identifier for each financial transaction, ensuring each record represents a distinct transaction.
- 'Project_Detail_ID' (INTEGER FOREIGN KEY) - References the Projects table to link each financial transaction to a specific project.
- 'Credit_Status_ID' (INTEGER FOREIGN KEY) - References the Credit_Status_Details table to establish a connection between the status of the credit.
- 'Original_Principal_Amount' (FLOAT NOT NULL) - The original US dollar amount of the loan that is committed and approved.
- 'Disbursed_Amount' (FLOAT NOT NULL) - The amount that has been disbursed from a loan commitment in equivalent US dollars, calculated at the exchange rate on the value date of the individual disbursements.
- 'Repaid_to_IDA' (FLOAT NOT NULL) - Total principal amounts paid or prepaid to IDA in US dollars, calculated at the exchange rate on the value date of the individual repayments. Repaid to IDA amounts include amounts written off under the Multilateral Debt Relief Initiative (MDRI).

E. Borrowers Table

- 'Borrower_ID' (INTEGER NOT NULL PRIMARY KEY) - Unique identifier for each borrower, ensuring each record represents a distinct borrower entity.
- 'Borrower' (VARCHAR NOT NULL) - The representative of the borrower to which the Bank loan is made.
- 'Country_ID' (INTEGER NOT NULL FOREIGN KEY) - References the Countries table to link each borrower to a specific country.
- 'Borrowers_Obligation' (FLOAT NOT NULL) - The Borrower Obligation is the outstanding balance for the loan as of the end of period date in US dollars equivalent. The Borrower's Obligation includes the amounts outstanding Due to 3rd parties.

VIII. Indexing

Without indexes, the database engine might resort to full table scans, which can be extremely slow, especially for large tables. With the appropriate indexes in place, the engine can quickly locate the necessary rows, reducing the need for scanning entire tables and significantly improving query performance.

Indexes store a sorted copy of the indexed columns, which reduces disk I/O operations during data retrieval. Instead of reading the entire table, the database engine can read only the index pages, resulting in faster query execution times.

A. Indexing on Query 1

Query | Query History

```
SELECT countries.Region, SUM(credit_transactions.Original_Principal_Amount) AS TotalPrincipalAmount,
SUM(credit_transactions.Disbursed_Amount) AS TotalDisbursedAmount
FROM credit_transactions
INNER JOIN project_details ON credit_transactions.Project_Detail_ID = project_details.Project_Detail_ID
INNER JOIN countries ON project_details.Country_ID = countries.Country_ID
GROUP BY countries.Region;

CREATE INDEX idx_query1_join ON credit_transactions (Project_Detail_ID);
CREATE INDEX idx_query1_project_detail_id ON project_details (Project_Detail_ID);
CREATE INDEX idx_query1_country_id ON project_details (Country_ID);
CREATE INDEX idx_query1_country_id_countries ON countries (Country_ID);
```

Fig. 2. Query 1 and the associated indices

Given below are some ways in which indexing has assisted with better performance of the query.

- *Faster Join Operations:* The index idx_query1_join on the Project_Detail_ID column of the credit_transactions table speeds up the join operation between credit_transactions and project_details. Instead of scanning the entire table, the database engine can quickly locate matching rows based on the indexed column.
- *Efficient Filtering:* Index idx_query1_country_id on the Country_ID column of the project_details table helps in filtering rows based on the Country_ID during the join operation. This improves the efficiency of the join process by reducing the number of rows that need to be processed.
- *Group By Optimization:* When performing the GROUP BY operation on the countries.Region column, the index idx_query1_country_id_countries on the Country_ID column of the countries table can assist in optimizing the grouping operation. Although it's not directly involved in the join, it can speed up the process of retrieving the regions for each country, which contributes to the overall performance of the query.

B. Indexing on Query 2

Query Query History

```sql
SELECT countries.Country, credit_status_details.Credit_Status,
COUNT(credit_status_details.Credit_Status) AS StatusCount
FROM credit_transactions
INNER JOIN project_details ON
credit_transactions.Project_Detail_ID = project_details.Project_Detail_ID
INNER JOIN countries ON project_details.Country_ID = countries.Country_ID
INNER JOIN credit_status_details ON
credit_transactions.Credit_Status_ID = credit_status_details.Credit_Status_ID
GROUP BY countries.Country, credit_status_details.Credit_Status;

CREATE INDEX idx_query2_join1 ON credit_transactions (Project_Detail_ID, Credit_Status_ID);
CREATE INDEX idx_query2_join2 ON project_details (Project_Detail_ID, Country_ID);
CREATE INDEX idx_query2_join3 ON countries (Country_ID);
CREATE INDEX idx_query2_credit_status_id ON credit_status_details (Credit_Status_ID);
```

Fig. 3. Query 2 and the associated indices

Given below are some ways in which indexing has assisted with better performance of the query.

- *Faster Join Operations:* The indexes idx_query2_join1 and idx_query2_join2 on the Project_Detail_ID column of the credit_transactions and project_details tables respectively, along with the Credit_Status_ID column in credit_transactions, help speed up the join operations. These indexes allow the database engine to quickly locate matching rows based on the indexed columns, reducing the time required for join operations.
- *Efficient Filtering:* The index idx_query2_join3 on the Country_ID column of the countries table improves the efficiency of filtering rows based on the Country_ID during the join operation between project_details and countries. This index ensures that the database engine can quickly identify the relevant country information needed for the query.
- *Group By Optimization:* The index idx_query2_credit_status_id on the Credit_Status_ID column of the credit_status_details table helps optimize the grouping process. By creating this index, the database engine can efficiently group rows based on the Credit_Status_ID, which contributes to faster query execution.

C. Query Performance Before Indexing

Query Query History

```sql
SELECT
query,
total_time:: Double precision
FROM pg_stat_statements
ORDER BY total_time DESC
LIMIT 4;
```

Data Output Messages Notifications

	query text	total_time double precision
1	SELECT countries.Region, SUM(credit_transactions.Original_Principal_Amount) AS T...	491.4924
2	SELECT countries.Country, credit_status_details.Credit_Status, COUNT(credit_status...	468.8968
3	SELECT countries.Country, SUM(credit_transactions.Original_Principal_Amount) AS ...	403.5861
4	SELECT credit_status_details.Credit_Status,	214.93529999999998

Total rows: 4 of 4 Query complete 00:00:00.122

Fig. 4. Performance Before Indexing

D. Query Performance After Indexing

Query Query History

```sql
SELECT
query,
total_time:: Double precision
FROM pg_stat_statements
ORDER BY total_time DESC
LIMIT 4;
```

Data Output Messages Notifications

	query text	total_time double precision
1	SELECT countries.Country, credit_status_details.Credit_Status, COUNT...	435.1263
2	SELECT countries.Country, SUM(credit_transactions.Original_Principal...	419.3526
3	SELECT countries.Region, SUM(credit_transactions.Original_Principal_...	393.4689
4	SELECT credit_status_details.Credit_Status,	221.1528

Total rows: 4 of 4 Query complete 00:00:00.256

Fig. 5. Performance After Indexing

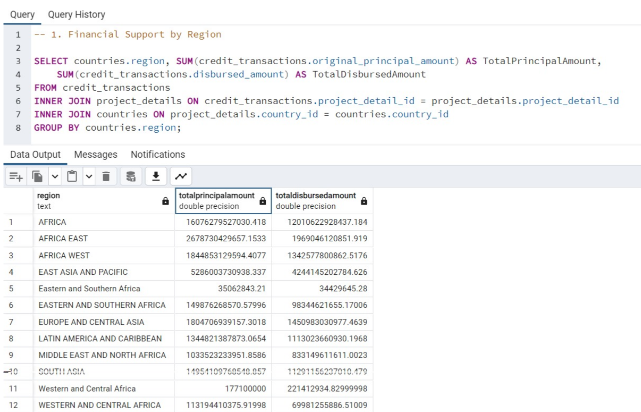
By examining the provided images, it's evident that prior to indexing, Query 1 required 491 milliseconds to execute, while Query 2 took 468 milliseconds to return results.

Following the implementation of indexing, Query 1's execution time decreased to 393 milliseconds, and Query 2 now completes in 435 milliseconds.

These results indicate that indexing has effectively decreased the execution time of both queries.

IX. SQL Queries

A. Total Financial Support by Region



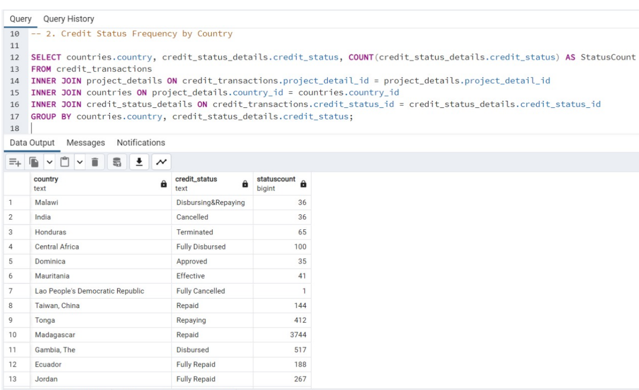
```sql
-- 1. Financial Support by Region

SELECT countries.region, SUM(credit_transactions.original_principal_amount) AS TotalPrincipalAmount,
    SUM(credit_transactions.disbursed_amount) AS TotalDisbursedAmount
FROM credit_transactions
INNER JOIN project_details ON credit_transactions.project_detail_id = project_details.project_detail_id
INNER JOIN countries ON project_details.country_id = countries.country_id
GROUP BY countries.region;
```

	region text	totalprincipalamount double precision	totaldisbursedamount double precision
1	AFRICA	16076279527030.418	12010622928437.184
2	AFRICA EAST	2678730429657.1533	1969046120851.919
3	AFRICA WEST	1844853129594.4077	1342577800862.5176
4	EAST ASIA AND PACIFIC	5286003730938.337	4244145202784.626
5	Eastern and Southern Africa	35062843.21	34429645.28
6	EASTERN AND SOUTHERN AFRICA	149876268570.57996	98344621655.17006
7	EUROPE AND CENTRAL ASIA	1804706939157.3018	1450983030977.4639
8	LATIN AMERICA AND CARIBBEAN	1344821387873.0654	1113023660930.1968
9	MIDDLE EAST AND NORTH AFRICA	1033523233951.8586	833149611611.0023
10	SOUTH ASIA	14954109768548.857	11291156237010.479
11	Western and Central Africa	177100000	221412934.82999998
12	WESTERN AND CENTRAL AFRICA	113194410375.91998	69981255886.51009

Fig. 6. Query1 Total Financial Support by Region

This query sums up the original principal amounts and disbursed amounts by region to understand the financial support distribution.

B. Credit Status Distribution by Country



```sql
-- 2. Credit Status Frequency by Country

SELECT countries.country, credit_status_details.credit_status, COUNT(credit_status_details.credit_status) AS StatusCount
FROM credit_transactions
INNER JOIN project_details ON credit_transactions.project_detail_id = project_details.project_detail_id
INNER JOIN countries ON project_details.country_id = countries.country_id
INNER JOIN credit_status_details ON credit_transactions.credit_status_id = credit_status_details.credit_status_id
GROUP BY countries.country, credit_status_details.credit_status;
```

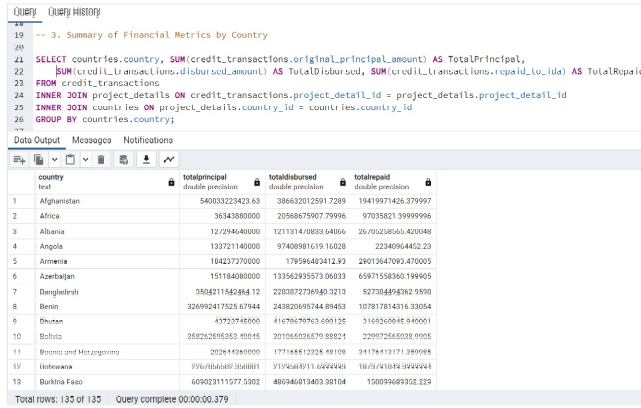
	country text	credit_status text	statuscount bigint
1	Malawi	Disbursing&Repaying	36
2	India	Cancelled	36
3	Honduras	Terminated	65
4	Central Africa	Fully Disbursed	100
5	Dominica	Approved	35
6	Mauritania	Effective	41
7	Lao People's Democratic Republic	Fully Cancelled	1
8	Taiwan, China	Repaid	144
9	Tonga	Repaying	412
10	Madagascar	Repaid	3744
11	Gambia, The	Disbursed	517
12	Ecuador	Fully Repaid	188
13	Jordan	Fully Repaid	267

Total rows: 1000 of 1026 Query complete 00:00:00.464

Fig. 7. Query2 Credit Status Distribution by Country

This query counts the number of projects in each credit status category for every country, providing an overview of credit management.

C. Total Repaid vs. Due Amounts by Country



```sql
-- 3. Summary of Financial Metrics by Country

SELECT countries.country, SUM(credit_transactions.original_principal_amount) AS TotalPrincipal,
    SUM(credit_transactions.disbursed_amount) AS TotalDisbursed, SUM(credit_transactions.repaid_to_ida) AS TotalRepaid
FROM credit_transactions
INNER JOIN project_details ON credit_transactions.project_detail_id = project_details.project_detail_id
INNER JOIN countries ON project_details.country_id = countries.country_id
GROUP BY countries.country;
```

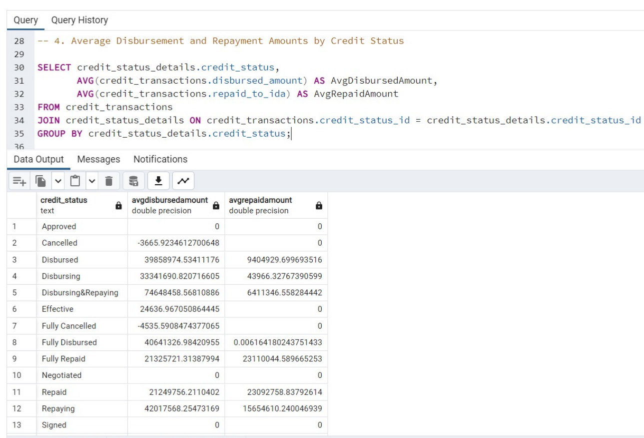
	country text	totalprincipal double precision	totaldisbursed double precision	totalrepaid double precision
1	Afghanistan	540033223423.63	386632012591.7289	19419971426.379997
2	Africa	36343880000	20568675907.79996	97035821.39999996
3	Albania	127294640000	121131470833.64066	26705258565.420048
4	Angola	133721140000	97408981619.16028	22340964452.23
5	Armenia	184237370000	179596403412.93	29013647093.470005
6	Azerbaijan	151184080000	133562935573.06033	65971558360.199905
7	Bangladesh	3504211542464.12	2283872736940.3213	527384494362.9598
8	Benin	326992417525.67944	243820695744.89453	107817814316.33054
9	Bhutan	43723745000	41670679763.690125	3169260045.940001
10	Bolivia	358262595353.40045	301065036579.88824	229972565038.9905
11	Bosnia and Herzegovina	202644360000	177165512325.48108	34176413171.359985
12	Botswana	[illegible]	[illegible]	[illegible]
13	Burkina Faso	609023111577.5302	486946813403.38104	150099689352.229

Total rows: 135 of 135 Query complete 00:00:00.379

Fig. 8. Query3 Total Repaid vs. Due Amounts by Country

This query compares the total amounts repaid to IDA with the total amounts due, offering insights into the repayment performance of different countries.

D. Average Disbursement and Repayment Amounts by Credit Status



```sql
-- 4. Average Disbursement and Repayment Amounts by Credit Status

SELECT credit_status_details.credit_status,
       AVG(credit_transactions.disbursed_amount) AS AvgDisbursedAmount,
       AVG(credit_transactions.repaid_to_ida) AS AvgRepaidAmount
FROM credit_transactions
JOIN credit_status_details ON credit_transactions.credit_status_id = credit_status_details.credit_status_id
GROUP BY credit_status_details.credit_status;
```

	credit_status text	avgdisbursedamount double precision	avgrepaidamount double precision
1	Approved	0	0
2	Cancelled	-3665.9234612700648	0
3	Disbursed	39858974.53411176	9404929.699693516
4	Disbursing	33341690.820716605	43966.32767390599
5	Disbursing&Repaying	74648458.56810886	6411346.558284442
6	Effective	24636.967050864445	0
7	Fully Cancelled	-4535.5908474377065	0
8	Fully Disbursed	40641326.98420955	0.006164180243751433
9	Fully Repaid	21325721.31387994	23110044.589665253
10	Negotiated	0	0
11	Repaid	21249756.2110402	23092758.83792614
12	Repaying	42017568.25473169	15654610.240046939
13	Signed	0	0

Total rows: 14 of 14 Query complete 00:00:00.260

Fig. 9. Query4 Avg Disbursement and Repayment Amounts by Credit Status

This query retrieves the average disbursed amount and the average amount repaid to IDA for each distinct credit status, providing insights into the financial aspects of different credit statuses.

E. Overview of Project Status in Kenya

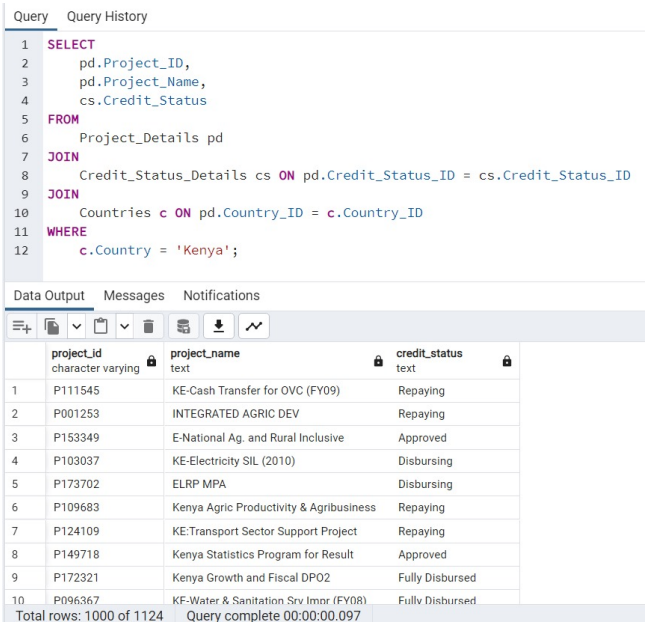


Fig. 10. Query5 Overview of Project Status in Kenya

This query provides insight into various projects and their respective credit statuses within that country.

F. Insertion of a new credit status record

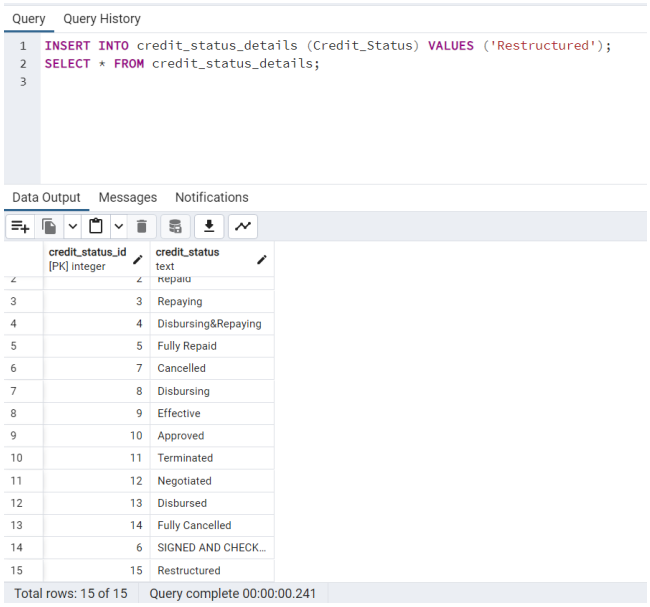


Fig. 11. Query6 Insertion and Selection of Indian Country Records

This query shows that a new record is inserted into the 'Credit_Status_Details' and now we have 15 entries in the table.

G. Insertion of a new project

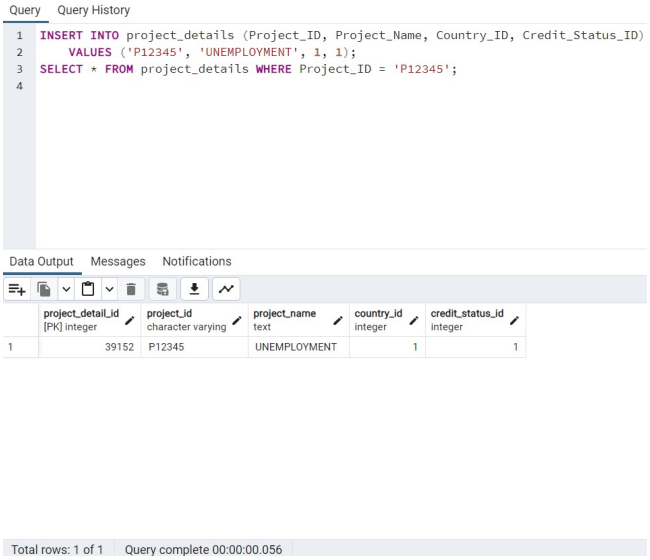


Fig. 12. Query7 Insertion of a new project record

We are inserting a new record into the 'Project_Details' table and we are using the select statement to display that the insertion was successful.

H. Updating the credit status information

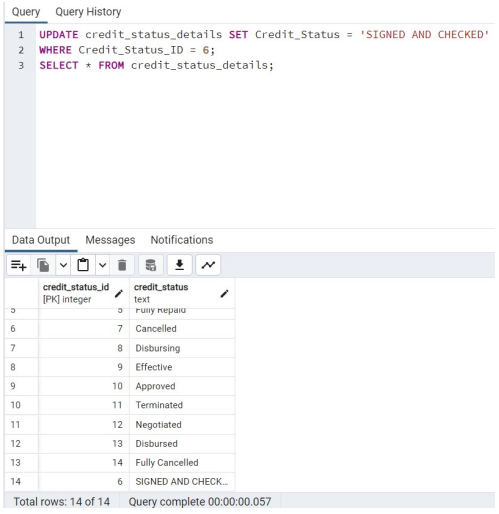


Fig. 13. Updating the credit status information

We are updating a record in the 'Credit_Status_Details' table for a particular entry and using the select statement to display successful update.

I. Deleting a record from Credit_Status_Details

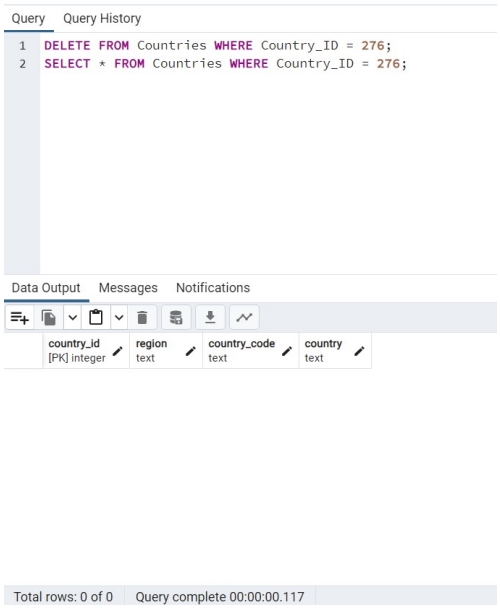


Fig. 14. Deletion of a record

We are deleting the newly inserted record from the 'Credit_Status_Details' table. Using the select statement we are verifying that the deleted record is not present in the table,

X. Problematic Queries

A. Join Without Indexes

- Problem Description: If a join operation does not use indexed columns, the database must perform a full table scan on one or both tables, leading to poor performance on large datasets.
- Potential Issue: If the columns Project_Details.Country_ID and Project_Details. Credit_Status_ID are not indexed, the query will require full table scans.
- Improvise Query:
  - Use Indexes: Usage of appropriate indexes on the columns involved in the joins and where clause. Given this query, we can create index Project_Details and Credit_Status_Details on Credit_Status_ID, as well as Project_Details and Countries on Country_ID.
  - Query Refactoring: Query can be rewritten for better performance.

B. Inefficient Aggregation

- Problem Description: Aggregations over large datasets without proper indexing can be slow, especially if the GROUP BY column isn't indexed.
- Potential Issue: If there's no index on Credit_Transactions. Project_Detail_ID and Project_Details. Country_ID, the query could be slow due to the lack of efficient grouping and joining.

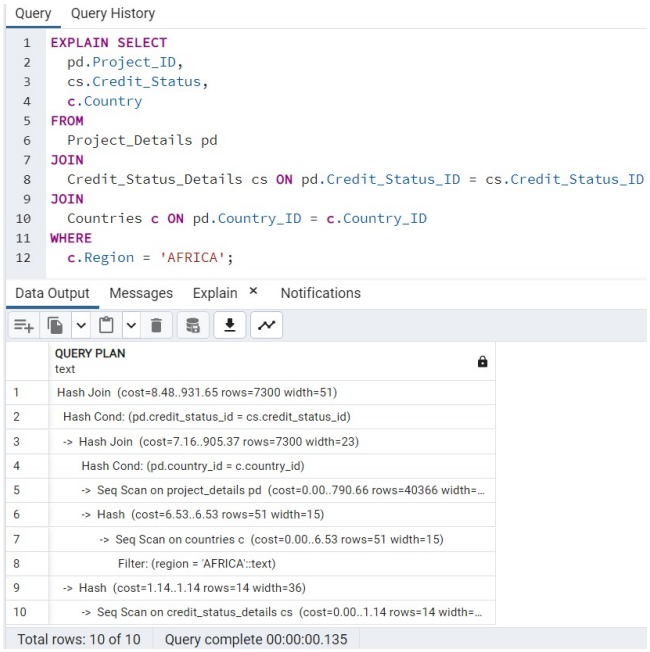


Fig. 15. Join Without Indexes

- Improvise Query:
  - Index on Join Columns: The Project_Detail_ID in Credit_Transactions and Country_ID in Project_Details are used for joins. Indexing should be done on these columns.
  - Index on Group By Column: Since the query uses GROUP BY c.Country, an index on Countries.Country will speed up the group aggregation.
  - Query Refactoring: Query can be rewritten for better performance.

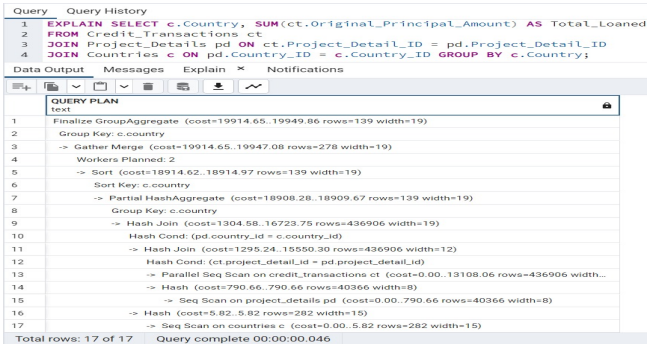


Fig. 16. Inefficient Aggregation

C. Subquery with Poor Performance

- Problem Description: Subqueries can sometimes lead to inefficient query plans, especially if they are not well-optimized by the query planner.
- Potential Issue: The subqueries might cause the database to execute the inner query multiple times, once for each

row processed by the outer query, which can be very slow if the subquery and outer query both scan large tables.

- Improvise Query:
  - Rewrite the Subquery as a Join: Rewrite the subquery using a JOIN, which will be more performant as it allows the database to optimize the execution plan better.
  - Use EXISTS Instead of IN: For a subquery, using EXISTS instead of IN return a list of IDs.
  - Indexes on Filtered and Joined Columns: Columns involved in the joins and WHERE clause will be indexed.

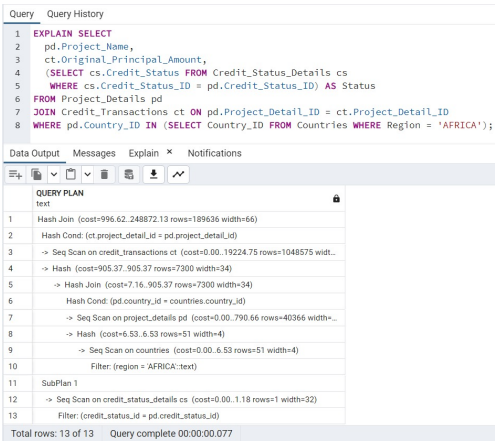


Fig. 17. Subquery with Poor Performance

XI. Dataset Source

The World Bank, IDA Statement Of Credits and Grants - Historical Data **dataset**

XII. Deployment

A running **website**