CanvasCart: Buy-and-Sell platform for Artworks

Vindhya Jain, Avanti Mittal, Khushi Bhardwaj

Introduction

CanvasCart is a buy-and-sell platform for artworks. This platforms allows artists to list their artworks for sale and buyers to browse, purchase, and manage their orders. Through the course of this assignment, we have written some common SQL queries that our database should handle, work on optimizing these queries and analyze execution/evaluation time before and after optimizing. Link to Colab Notebook.

SQL Queries

1. Extract a list of all <artist_name, artwork> who have artwork listings in all months in 2023

```sql
SELECT u.FirstName AS ArtistFirstName, u.LastName AS ArtistLastName, art.
    Title AS ArtworkTitle
FROM User u,  Artwork art
WHERE u.UserID = art.ArtistID AND strftime('%Y', art.DateAdded) = '2023'
```

output:

```
Artist: Benjamin Hicks, Artwork: Reason.
Artist: Benjamin Hicks, Artwork: Use list market father.
Artist: Megan Booker, Artwork: Already outside actually.
Artist: Megan Booker, Artwork: Capital style occur.
Artist: Michelle Williams, Artwork: Pretty.
Artist: Michelle Williams, Artwork: Safe often way national.
Artist: Julie Campbell, Artwork: Around Republican.
Artist: Johnny Velasquez, Artwork: Far born.
```

2. From the above list, print the names of all artists who have at least one sculpture

```sql
SELECT u.FirstName, u.LastName
FROM User u, Artwork art
WHERE u.UserID = art.ArtistID AND strftime('%Y', art.DateAdded) = '2023' AND
     art.medium = 'Sculpture'
```

output:

```
Artists with at least one sculpture:
Johnny Velasquez
```

3. Extract a list of all <artist_profile> information for whom the database does not have any artwork listing

```sql
SELECT u.UserID, u.FirstName, u.LastName, a.Biography, a.PortfolioURL
FROM User u, Artist a , Artwork art
WHERE a.ArtistID = art.ArtistID AND u.UserID = art.ArtistID AND art.ArtistID
     IS NULL
```

output:

```
Artists with no artwork listings:
+----------+-------------+-----------+-----------------------------------------------------------------------------------------------------------------------+-----------------------------------+
|   UserID | FirstName   | LastName  | Biography                                                                                                             | Portfolio URL                     |
+==========+=============+===========+=======================================================================================================================+===================================+
|        6 | Zachary     | Peters    | Particularly travel reveal. Able husband total beat particular available.                                             | https://www.schultz-washington.c  |
|          |             |           | Week really raise home let. Require former else painting.                                                             |                                   |
+----------+-------------+-----------+-----------------------------------------------------------------------------------------------------------------------+-----------------------------------+
|        8 | Carol       | Ford      | Expert laugh write want race range should. Large safe radio fill institution.                                         | https://www.reyes.com/            |
|          |             |           | Special hour notice life place. Increase sell tend which relate kitchen. Child cause thank raise it behavior.        |                                   |
+----------+-------------+-----------+-----------------------------------------------------------------------------------------------------------------------+-----------------------------------+
```

4. Print a list of all buyers who have made purchases of oil paintings in 2022

```sql
SELECT DISTINCT u.UserID, u.FirstName, u.LastName
FROM Orders o,Item i, Artwork art, User u
WHERE u.UserID = o.BuyerID AND o.OrderID = i.OrderID AND i.ArtworkID = art.
    ArtworkID AND art.Medium = 'Oil' AND strftime('%Y', o.OrderDate) = '2022'
```

output:

```
Buyers who purchased oil paintings in 2022:
BuyerID: 20, Name: Pam Hicks
BuyerID: 31, Name: Lisa Haynes
BuyerID: 40, Name: Lauren Rowe
BuyerID: 18, Name: Lisa Salazar
BuyerID: 26, Name: Amy Miller
BuyerID: 46, Name: Zachary Lee
BuyerID: 49, Name: Angela Daniel
BuyerID: 20, Name: Pam Hicks
BuyerID: 40, Name: Lauren Rowe
BuyerID: 31, Name: Lisa Haynes
BuyerID: 31, Name: Lisa Haynes
```

5. From the above list, derive a list of the artists and their profile information

```sql
SELECT DISTINCT u.FirstName AS ArtistFirstName, u.LastName AS ArtistLastName
    , art.Biography, art.PortfolioURL
FROM User u, Artist art, Artwork a, Item i, Orders o
WHERE u.UserID = art.ArtistID AND art.ArtistID = a.ArtistID AND a.ArtworkID
    = i.ArtworkID AND i.OrderID = o.OrderID AND a.Medium = 'Oil' AND strftime
    ('%Y', o.OrderDate) = '2022'
```

output:

Artist First Name	Artist Last Name	Biography	Portfolio URL
Cory	Kent	Light whose nation election. Already training act point impact tend. Agreement eye policy. Where less common wish mind number can.	http://www.orozco.info/
Joshua	Allen	Say your including time.	https://campbell.org/
		Yet effort under. Boy staff religious ready find night. Effect skill never. Budget week knowledge tax.	
Jorge	Patterson	Standard something thought society possible build. Without goal thing age us her. Send car yard experience.	http://price.biz/
		Summer business record strategy eight. Fact information eight ok laugh.	
Jessica	Jones	Stop foot animal nature view old. Involve seat growth in of.	https://ball.com/
		Member account former not. Its contain build should. Bit meet day. Must economy rule couple white capital suddenly.	
Brian	Edwards	Skin drop change bar. Try environment audience since accept kitchen cup. Others section its almost there child save.	http://www.flores.com/
Heather	Dixon	House subject weight citizen front him hospital. Property cell well instead free. Offer morning management modern clearly modern. Imagine animal others.	https://smith-fox.com/

6. Derive a list of all <buyer_profiles> who have not made any purchases

```sql
SELECT b.BuyerID, u.FirstName, u.LastName, b.ShippingAddress, b.Membership
FROM Buyer b
JOIN User u ON b.BuyerID = u.UserID
LEFT JOIN Orders o ON b.BuyerID = o.BuyerID
WHERE o.OrderID IS NULL
```

output:

```
Buyers who have not made any purchases:
```

BuyerID	FirstName	LastName	ShippingAddress	Membership
13	Anita	Reynolds	69757 Long Union Mcculloughfurt, NH 64467	0
15	Jennifer	Little	706 Rachel Walks Jonathanborough, OK 13724	1
18	Ashley	Woods	152 Tracey Valley West Bobbystad, AR 67339	1
20	Mark	Montgomery	Unit 2018 Box 7531 DPO AA 18248	0
22	Timothy	Oconnor	623 Rhonda Highway Apt. 983 South Alicia, IL 14100	1
29	David	Gonzalez	891 Christopher Walk Vargasborough, MO 90068	0
37	Veronica	Anderson	1278 Mclaughlin Vista Port Leslie, AK 35947	1
41	Gwendolyn	Dixon	5854 Warren Track Suite 086 Philipchester, FM 55478	1
42	Robert	Stewart	74161 Larson Walk Suite 236 Lake David, MD 56975	1
48	Gabriel	Mcdonald	86470 Brittany Plains Suite 399 Robertview, KS 68498	1
49	Alicia	Hill	440 Kaitlin Islands Suite 762 North Anne, KY 91324	0

Optimizing the above queries

Query optimization is the process of improving the efficiency of SQL queries by transforming them into more efficient forms without changing their output. It involves techniques like rearranging query operations, applying conditions early, and replacing inefficient operations with more effective ones, all aimed at reducing execution time and resource consumption.

The following are the steps through which the query tree can be optimised

a. Initial query tree for SQL query

b. Moving select operations down the tree

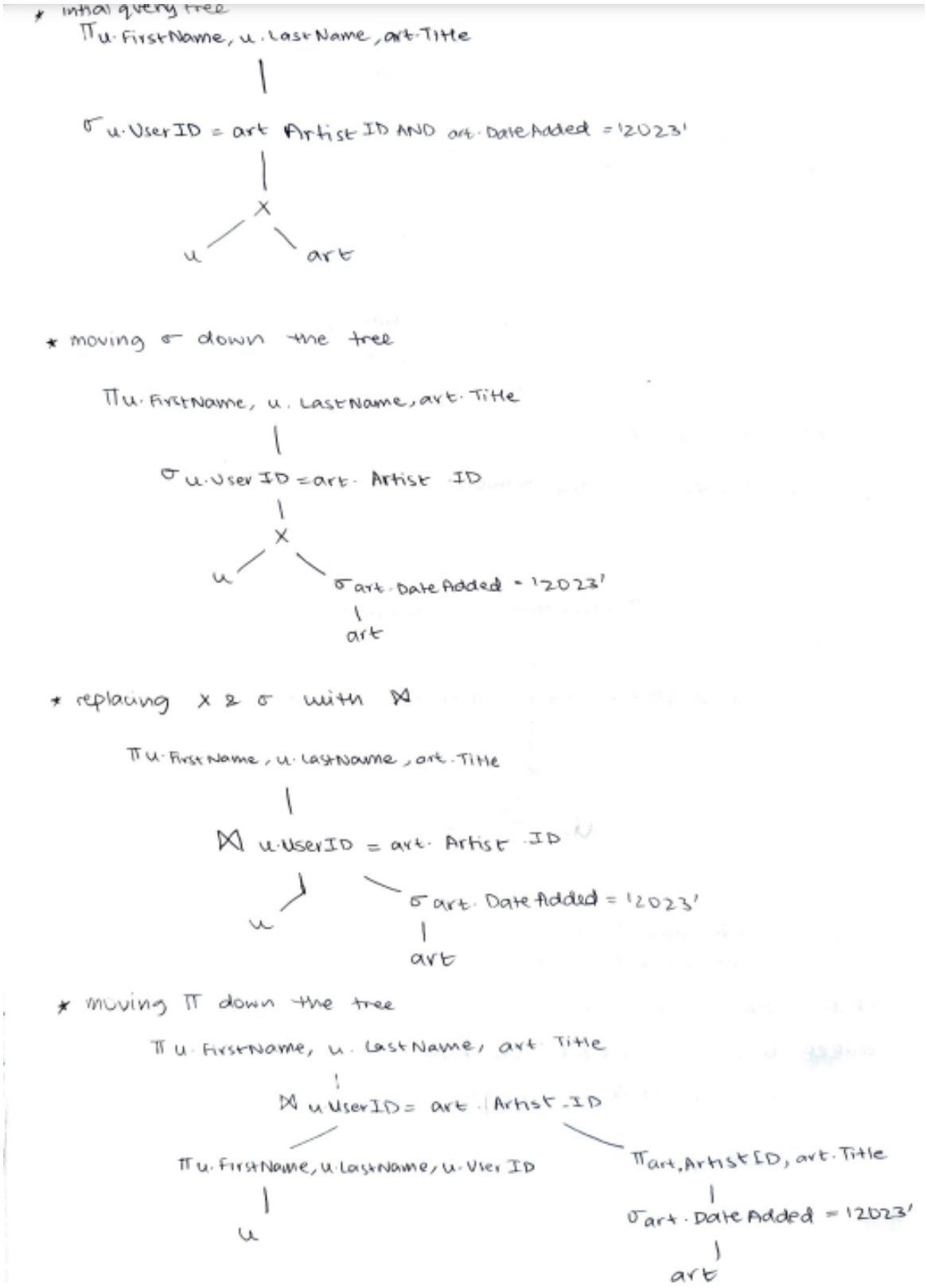
c. Applying the more restrictive select options first

d. Replacing cartesian product and select with join operations

e. Moving project operations down the query tree

Each of the above queries are optimised via this approach (as shown below).

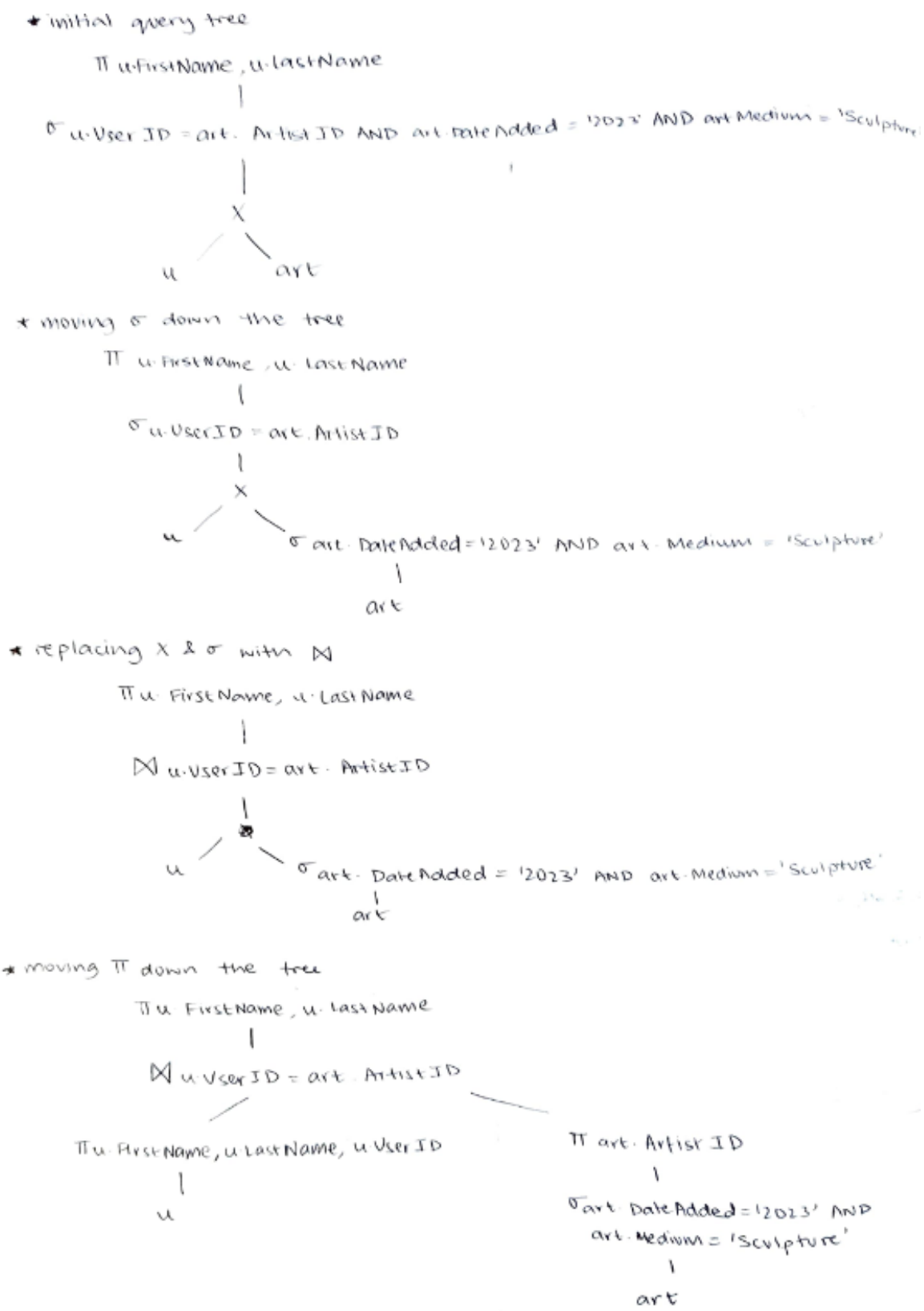
1. Extract a list of all <artist_name, artwork> who have artwork listings in all months in 2023.



Optimised Relational Algebra:

$$\pi_{u.FirstName,\ u.LastName,\ art.Title}((\pi_{u.FirstName,\ u.LastName,\ u.UserID}(u)) \bowtie_{u.UserID=art.ArtistID} (\pi_{art.ArtistID,\ art.Title}(\sigma_{art.DateAdded\ =\ '2023'}(art))))$$

2. From the above list, print the names of all artists who have at least one sculpture.



Optimised Relational Algebra:

$$\pi_{u.FirstName,\ u.LastName}((\pi_{u.FirstName,\ u.LastName,\ u.UserID}(u)) \bowtie_{u.UserID=art.ArtistID} (\pi_{art.ArtistID}(\sigma_{art.DateAdded\ =\ '2023'\ AND\ art.Medium\ =\ 'Sculpture'}(art)))$$

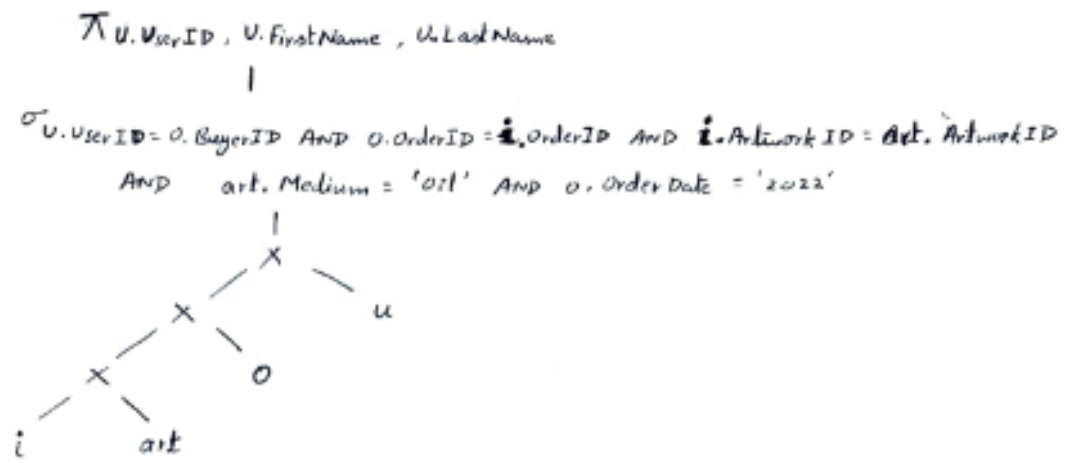
3. Extract a list of all <artist_profile> information for whom the database does not have any artwork listing

Optimised Relational Algebra:

$\pi_{u,UserID,\ u.FirstName,\ u.LastName,\ a.Biography,\ a.Portfolio}((\sigma_{art.ArtworkID=NULL}((a)\bowtie_{a.ArtistID=art.ArtistID}(\pi_{art.ArtistID,\ art.ArtworkID}(art)))) \bowtie_{u.UserID=art.ArtistID} (\pi_{u.UserID,\ u.FirstName,\ u.LastName}(u))$

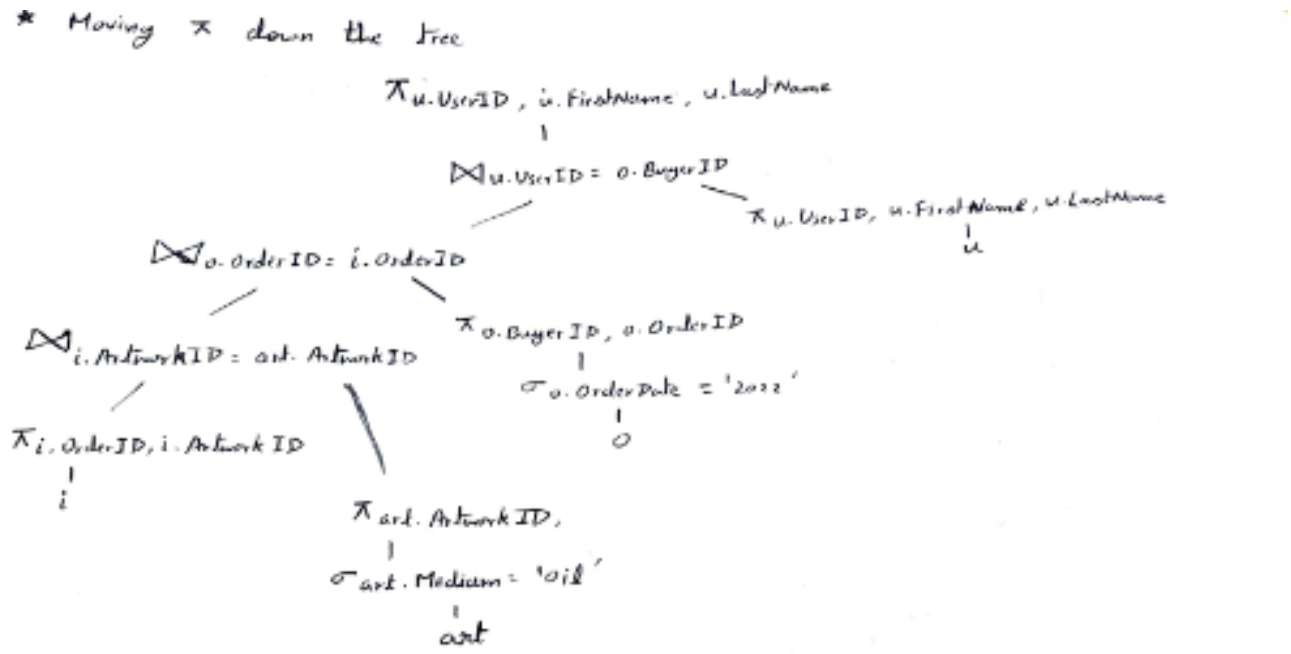
4. Print a list of all buyers who have made purchases of oil paintings in 2022.

* Initial query tree



* Moving (SELECT) σ down the tree

* Replacing σ & × with ⋈

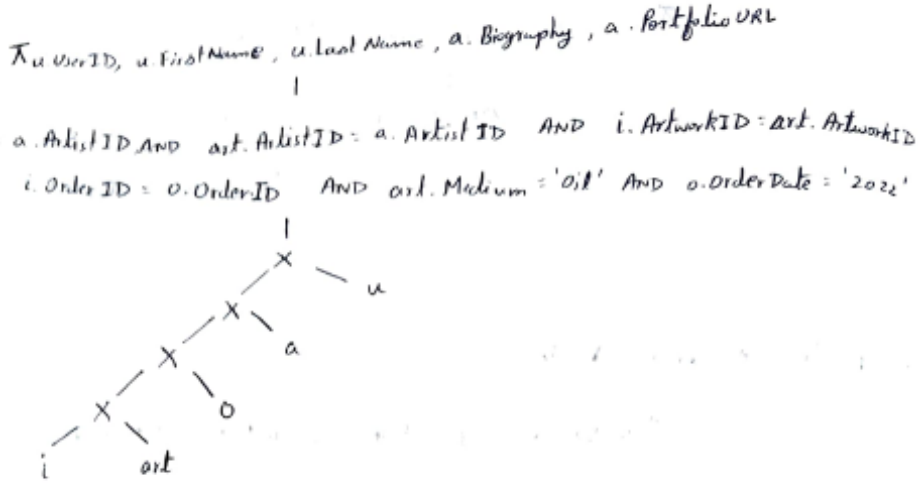


Optimised Relational Algebra:

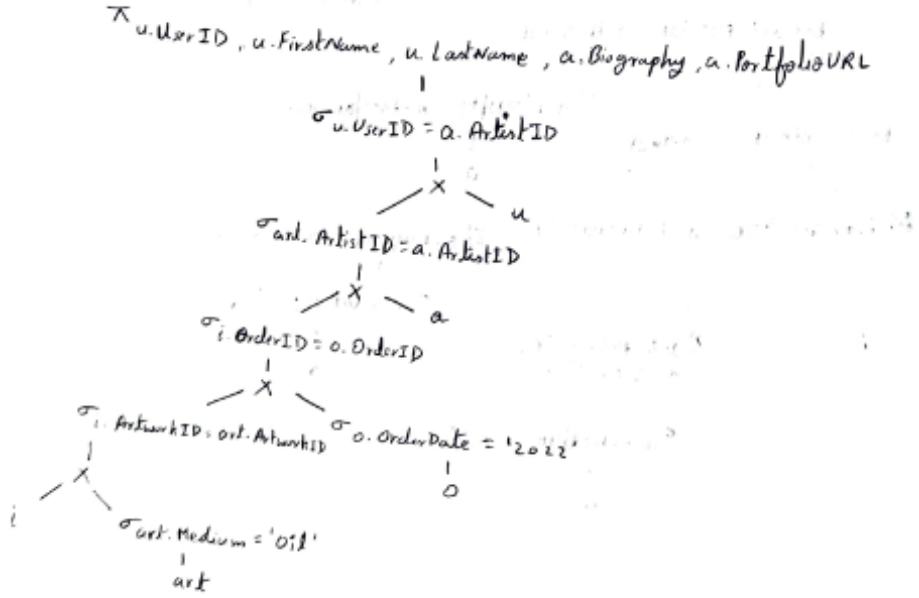
$$\pi_{u,UserID,\ u.FirstName,\ u.LastName}((((\pi_{i.OrderID,\ ArtworkID}(i)) \bowtie_{i.ArtworkID=art.ArtworkID} (\pi_{art.ArtworkID}(\sigma_{art.Medium='oil'}(art)))) \bowtie_{o.OrderID=i.OrderID} (\pi_{o.BuyerID,\ o.OrderID} \sigma_{o.OrderDate='2022'}(o)))) \bowtie_{u.UserID=o.BuyerID} (\pi_{u.UserID,\ u.FirstName,\ u.LastName}(u)))$$

5. From the above list, derive a list of the artists and their profile information.

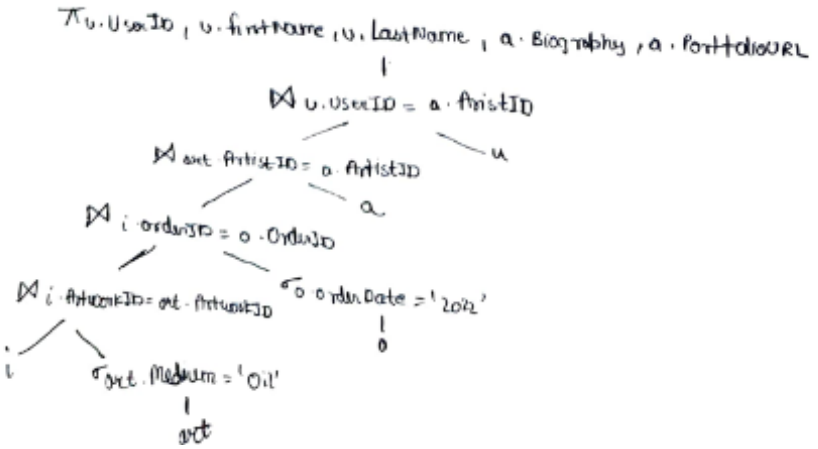
* Initial Query Tree



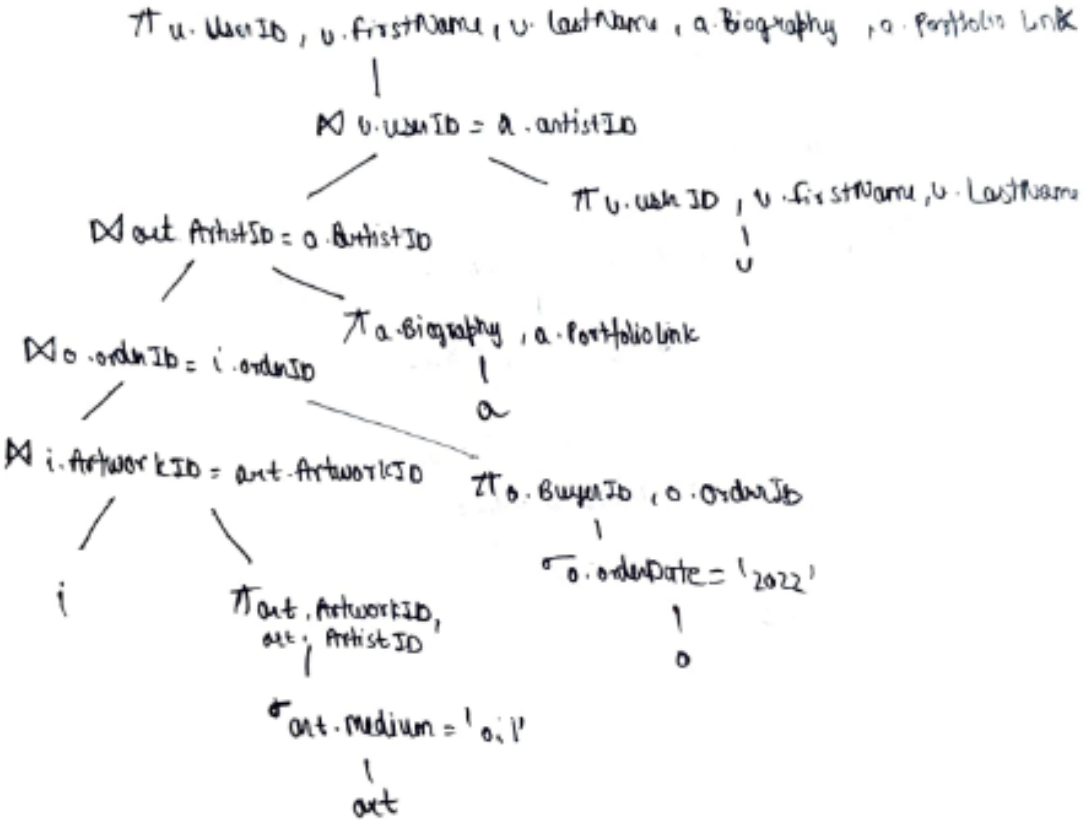
* Moving σ down the tree



* Replacing σ & × with ⋈

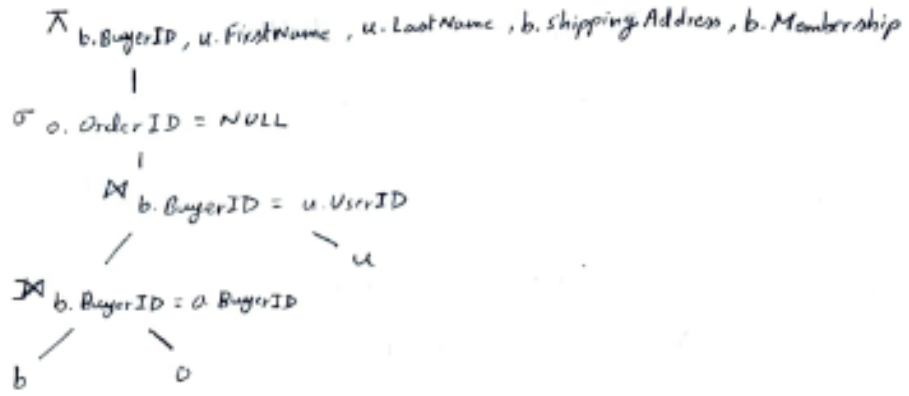


* moving π down the tree

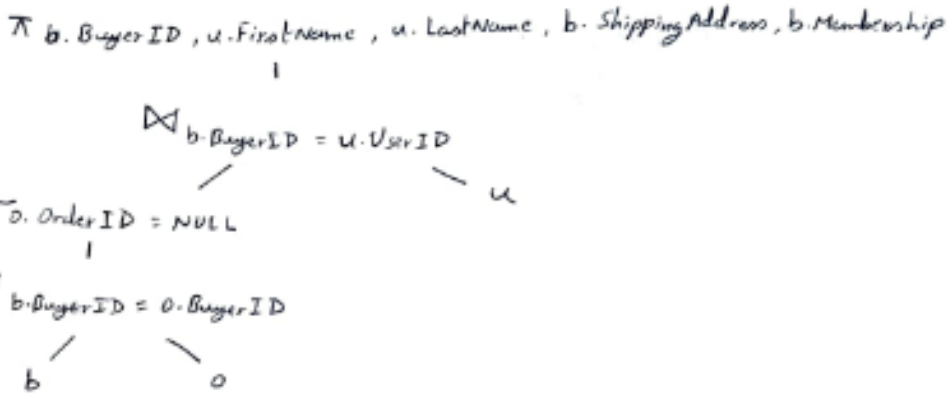


6. Derive a list of all <buyer_profiles> who have not made any purchases.

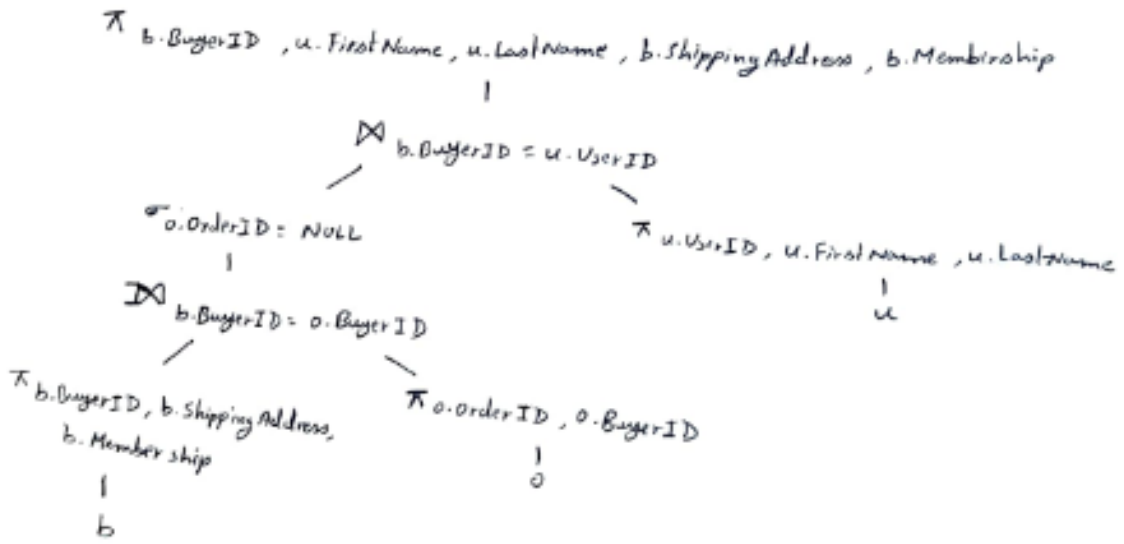
* Initial Query Tree



* Moving σ down the tree



* Moving π down the tree



Optimised Relational Algebra:

$\pi_{b.BuyerID,\ u.FirstName,\ uLastName,\ b.ShippingAddress,\ b.Membership}((\sigma_{o.OrderID=NULL}((\pi_{b.BuyerID,\ b.ShippingAddress,\ b.Membership}(b))\bowtie_{B]b.BuyerID=o.BuyerID}(\pi_{o.OrderID,\ o.BuyerID}(o))))\bowtie_{u.UserID,\ u.FirstName,\ u.LastName}(\pi_{u.UserID,\ u.FirstName,\ u.LastName}(u)))$

Equivalent SQL queries

1. Extract a list of all <artist_name, artwork> who have artwork listings in all months in 2023.

```
SELECT u.FirstName, u.LastName, art.Title
FROM (
    SELECT u.FirstName, u.LastName, u.UserID
    FROM User u
) u
JOIN (
    SELECT art.ArtistID, art.Title
    FROM Artwork art
    WHERE strftime('%Y', art.DateAdded) = '2023'
) art
ON u.UserID = art.ArtistID;
```

2. From the above list, print the names of all artists who have at least one sculpture.

```
SELECT u.FirstName, u.LastName, art.Title
FROM (
    SELECT u.FirstName, u.LastName, u.UserID
    FROM User u
) u
JOIN (
    SELECT art.ArtistID, art.Title
    FROM Artwork art
    WHERE strftime('%Y', art.DateAdded) = '2023'
) art
ON u.UserID = art.ArtistID;
```

3. Extract a list of all <artist_profile> information for whom the database does not have any artwork listing

```
SELECT u.UserID, u.FirstName, u.LastName, a_wo_art.Biography, a_wo_art.
   PortfolioURL
FROM (
    SELECT*
FROM Artist a
LEFT JOIN (
            SELECT art.ArtistID, art.ArtworkID
            FROM Artwork art
        )art
ON art.ArtistID = a.ArtistID
WHERE art.ArtworkID IS NULL ) AS a_wo_ART
JOIN (
    SELECT u.UserID, u.FirstName, u.LastName
    FROM User u
) u
ON u.UserID = a_wo_art.ArtistID;
```

4. Print a list of all buyers who have made purchases of oil paintings in 2022.

```
SELECT
    u.UserID,
    u.FirstName,
    u.LastName
FROM
    (SELECT i.OrderID, i.ArtworkID
    FROM Item i) i
JOIN
    (SELECT art.ArtworkID
```

```
    FROM Artwork art
    WHERE art.Medium = 'Oil') art
ON i.ArtworkID = art.ArtworkID
JOIN
    (SELECT o.OrderID, o.BuyerID
    FROM Orders o
    WHERE strftime('%Y', o.OrderDate) = '2022') o
ON i.OrderID = o.OrderID
JOIN
    (SELECT  u.UserID, u.FirstName, u.LastName
    FROM User u) u
ON o.BuyerID = u.UserID;
```

5. From the above list, derive a list of the artists and their profile information.

```
SELECT DISTINCT
    u.UserID,
    u.FirstName,
    u.LastName,
    a.biography,
    a.portfolioURL
FROM
    (SELECT i.OrderID, i.ArtworkID
    FROM Item i) AS i
JOIN
    (SELECT art.ArtworkID, art.ArtistID
    FROM Artwork art
    WHERE art.Medium = 'Oil') AS art
ON i.ArtworkID = art.ArtworkID
JOIN
    (SELECT o.OrderID, o.BuyerID
    FROM Orders o
    WHERE strftime('%Y', o.OrderDate) = '2022') AS o
ON i.OrderID = o.OrderID
JOIN
    (SELECT a.ArtistID, a.biography, a.portfolioURL
    FROM Artist a) AS a
ON art.ArtistID = a.ArtistID
JOIN
    (SELECT u.UserID, u.FirstName, u.LastName
    FROM User u) AS u
ON a.ArtistID = u.UserID;
```

6. Derive a list of all <buyer_profiles> who have not made any purchases.

```
SELECT  b_wo_o.BuyerID, u.FirstName, u.LastName,  b_wo_o.ShippingAddress,
   b_wo_o.Membership
FROM (
    SELECT *
    FROM Buyer b
    LEFT JOIN (SELECT o.OrderID, o.BuyerID
                FROM Orders o
            ) o
    ON b.BuyerID=o.BuyerID
    WHERE o.OrderID IS NULL) AS b_wo_o
JOIN(
    SELECT u.UserID, u.FirstName, u.LastName
    FROM User u
) u
```

```
14 ON  b_wo_o.BuyerID = u.UserID;
```

Evaluation Time Analysis

Let us look at how much time (seconds) the initial and optimised query take to execute.

	Initial	*Optimised*
Query 1	0.000162	0.000129
Query 2	0.000118	0.000078
Query 3	0.000144	0.000112
Query 4	0.000207	0.000155
Query 5	0.000213	0.000151
Query 6	0.000213	0.000195

Clearly, the optimised queries take lesser time to execute.

Inter-Language Differences:
Since both queries were executed in the same SQL environment, inter-language differences were not a factor in this comparison. However, if the queries had been run in different database systems or interfaces, differences in their query optimization capabilities and execution engines could have impacted performance.

Intra-Language Differences:
The reduction in execution time can be attributed to several intra-language optimizations:

1. Rearranging SELECT Operations: Moving SELECT operations earlier in the query tree allowed for filtering out unnecessary records sooner, which reduced the size of intermediate results.
2. Replacing Cartesian Products with JOINs: Optimizing joins to replace Cartesian products minimized the number of tuple combinations and improved efficiency.
3. Pushing PROJECT Operations Down: Reducing the number of columns processed early in the query further decreased the data size and processing time.

In conclusion, the optimized queries demonstrated a substantial improvement in performance due to effective query restructuring and operation optimization. This resulted in faster execution times and more efficient data retrieval.

Search Algorithm in SQLite

The search algorithm primarily revolves around indexing and table scanning.

Table Scanning

- Full Table Scan: If no index is available or applicable for a query, SQLite performs a full table scan, where it examines each row one by one. This is the simplest but often the slowest method, especially for large tables.

Indexing
B-tree Index: SQLite uses B-tree structures for indexing. A B-tree is a balanced tree that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time. When a query is executed, SQLite checks if there's an index that can be used to speed up the search.

- Primary Key Indexes: Automatically created for primary keys, allowing quick lookups based on the primary key.
- Custom Indexes: We can create indexes on specific columns to make searches faster. For example, we created an index on the Medium column that would speed up searches for rows by Medium.

For a query like SELECT * FROM Artwork WHERE ArtworkID = OIL_01; SQLite will:

1. Check if there's an index on the ArtworkID column.
2. If an index exists, it will use the B-tree index to quickly locate the row with ArtworkID = OIL_01.
3. If no index exists, it will perform a full table scan to find the matching row.

Potential Affects of the SQLite Search Algorithm on our queries

The choice of search algorithm and indexing structure can significantly affect the execution time of SQL queries. Here's how:

1. **Query Speed**: With B-Tree indexing, searches for specific values or ranges become much faster because the index allows the database engine to quickly navigate to the relevant data rather than scanning the entire table. This results in faster query execution, especially for large datasets.
2. **Reduced Disk I/O**: Since B-Trees minimize the number of nodes accessed during a search, they reduce the amount of disk I/O. Each node access typically corresponds to a disk page read, so fewer accesses translate to less I/O overhead and faster query response times.
3. **Efficient Data Retrieval**: For queries involving range scans (e.g., finding records within a specific range), B-Tree indexes are particularly effective. They enable the database to efficiently locate the starting point and sequentially access the required records without scanning unrelated data.
4. **Update and Maintenance Costs**: While B-Tree indexes improve search performance, they also incur costs for insertions, updates, and deletions. Each modification may require rebalancing of the tree, which can impact performance. However, these costs are typically outweighed by the benefits for read-heavy workloads.
5. **Index Size and Memory Usage**: Indexes consume additional disk space and memory. The overhead of maintaining indexes must be balanced against the performance gains they provide. For very large datasets, proper indexing strategy and regular maintenance are crucial to avoid excessive resource consumption.

In summary, the search algorithm used by SQL packages, particularly B-Tree indexing, plays a vital role in optimizing query performance by speeding up data retrieval, reducing disk I/O, and efficiently managing data access. It's essential to consider the trade-offs related to index maintenance and resource usage to ensure optimal performance.

Compilation / interpretation strategy of C and Python languages

C is a compiled language, meaning that its source code is transformed into machine code by a compiler before execution.

Compilation Process:

1. Preprocessing: The source code undergoes preprocessing where directives like #include and #define are processed.
2. Compilation: The preprocessed code is then compiled into assembly language, which is a low-level representation of the code.
3. Assembly: The assembly code is converted into object code by an assembler.
4. Linking: The object code is linked with libraries and other modules to produce an executable file.
5. Execution: The resulting executable file is run directly by the operating system, leading to fast execution times.

Since C is compiled into machine code, it typically exhibits high performance and efficiency. The compilation step allows for various optimizations to be applied to the code. Errors are detected at compile-time, which can help catch issues before the program is executed.

Python is primarily an interpreted language, meaning that its source code is executed line-by-line by an interpreter at runtime.

Interpretation Process:

1. Source Code: The Python source code is read and executed by the Python interpreter.
2. Bytecode Compilation: Python source code is first compiled into bytecode, which is a low-level, platform-independent representation of the code.
3. Execution: The Python Virtual Machine (PVM) executes the bytecode. This involves interpreting the bytecode instructions and translating them into machine code at runtime.

Python code is executed in an interpreted environment, meaning that it's run by the interpreter directly from the source code or bytecode.This is why python tends to have slower execution times compared to compiled languages like C, due to the overhead of interpretation. However, it offers ease of development and flexibility. Errors are detected at runtime, which can sometimes make debugging more challenging compared to compile-time error detection in compiled languages.