

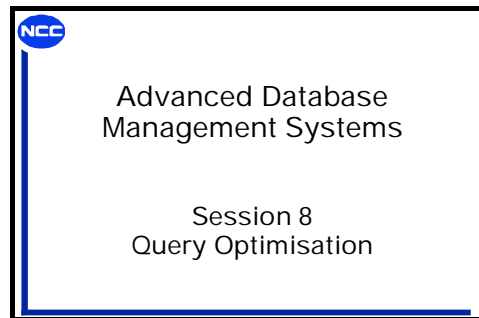
Session 8

Query Optimisation

1 Introduction

V8.1

(10 minutes)



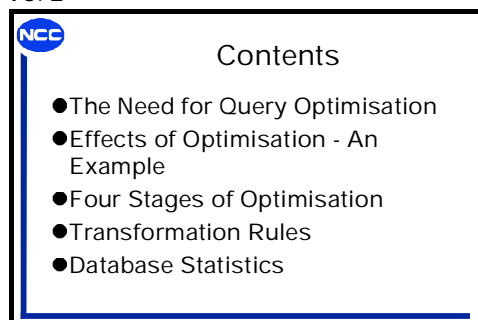
Query optimisation is an important component of a modern relational database system. When processing user queries, the query optimiser transforms them into a set of low level operations, and decides in what order these operations are to be executed in the most efficient way.

In a non-relational database system, any optimisation has to be carried out manually. Relational database systems, however, offer a system-managed optimisation facility by making use of the wealth of statistical information available to the system. The overall objective of a query optimiser is to improve the efficiency and performance of a relational database system.

Inform students that a handout containing a full set of visuals will be provided to them at the end of this lecture.

1.1 Summary of Topics to be Covered

V8.2



The topics detailed in the visual will be discussed during this session.

2 The Need for Query Optimisation

(10 minutes)

V8.3

NCC

The Need for Query Optimisation

- **Description**
 - **A program for efficient evaluation of relational queries**, making use of relevant statistic information
- **Objective**
 - **Choose the most efficient strategy for implementing** a given relational query, thereby improve system performance
- **Need**
 - **Perform automatic navigation**
 - **Achieve acceptable performance**
 - **Minimise existing discrepancies**

- *Description*

A query optimiser is essentially a program which is used for the efficient evaluation of relational queries, making use of relevant statistic information.

- *Objective*

The main objective of query optimisation is to choose the most efficient strategy for implementing a given relational query, and thereby improve on the system performance.

- *Need*

- To perform automatic navigation

A database system based on either the hierarchical model or network model requires users to navigate across the database to locate and retrieve the desired data. A relational database system, however, allows users simply to state what data they require and leave the system to locate that data in the database.

- To achieve acceptable performance

Although there are many different ways available to perform a single user query, this automatic navigation can be optimised by choosing and executing the most efficient plan, selected on the basis of system statistical information. By executing queries in the most cost-effective way, the overall system performance can be improved /enhanced.

- To minimise existing discrepancies

The speed discrepancy between CPU and I/O devices is still enormous despite rapid recent advances in computer technology. As an I/O access is one of the most costly and frequently performed operations in a typical database system, it is the main aim of the query optimiser to minimise the I/O activities by choosing the cheapest query plan for a given query, usually the one with a minimum number of tuple I/O operations.

3 Effects of Optimisation - An Example

(30 minutes)

V8.4

NCC

Effects of Optimisation - 1

Student = (Stud_No, Stud_Name, Gender, Address)
 Lending = (Lending_No, Stud_No, Book_No)
 Book = (Book_No, Title, Author, Edition)

In order to appreciate the effects of query optimisation on database performance, we use a simple example as an illustration.

Consider the following 3 tables: Student, Lending and Book. The attributes highlighted are the keys for the relevant relations shown in Visual V8.4.

Issue Handout 8.1 which shows the example given here.

V8.5

NCC

Effects of Optimisation - 2

Query: Retrieve the names of students who have borrowed the book B1

```
Select Distinct Student.Stud_Name
From Student, Lending
Where Student.Stud_No = Lending.Stud_No
And Lending.Book_No = 'B1'
```

- Suppose the database contains 100 students and 10,000 lendings, of which only 50 are for book B1
- Assume that results of up to 50 tuples can be kept in memory

Consider the following query:

Retrieve the names of students who have borrowed the book B1.

This query can be expressed in SQL as shown in Visual V8.5.

We further make the following two assumptions:

- The database contains 100 students and 10,000 lendings, of which only 50 are for book B1.
- It is possible to hold up to 50 tuples in memory, without having to write back to disk.

3.1 Query Execution Plan A - No Optimisation

V8.6

NCC

Query Execution Plan A - No Optimisation

Join-Select-Project

1. *Join* relations Student and Lending over Stud_No
2. *Select* the result of Step 1 for just the tuples for book B1
3. *Project* the result of Step 2 over Stud_Name to get result (50 max)

Total tuple I/O: 1,020,000

In this first approach, the operations required by the query are performed in the sequence:

Join-Select-Project

We calculate the number of database accesses (tuple I/O operations) occurred in each operation.

1. *Join* relations Student and Lending over Stud_no.
 - For each Student row, every Lending row will be retrieved and tested (reading each of the 100 Student rows 10,000 times);
 - Every Lending row will match with one Student row, so the number of joined rows in this intermediate relation is 10,000. These have to be written back to disk (only 50 tuples can be held in memory - see assumptions).

So, the number of tuple I/Os occurred in this step is:

$$(100 * 10,000) + 10,000 = 1,010,000$$

2. *Select* the result of Step 1 for just the tuples for book B1.
 - This results in reading the 10,000 joined tuples (obtained in step 1) back into memory.
 - Then Select produces a relation containing only 50 tuples, which can be kept in memory (see assumption).

The number of tuple I/Os in this step is:

$$10,000 + 0 = 10,000$$

3. *Project* the result of Step 2 over Stud_Name to get result (50 max).
 - This results in reading a relation of 50 tuples (obtained in step 2) which is already in memory, and producing a final relation of no more than 50 tuples, again in memory.

The number of tuple I/O in this step is:

$$0 + 0 = 0$$

Therefore, the total number of tuple I/Os for query plan A is:

$$(1,010,000 + 10,000).$$

Total tuple I/O: 1,020,000

3.2 Query Execution Plan B - With Optimisation

V8. 7

NCC Query Execution Plan B - With Optimisation

Select-Join-Project

1. *Select* the relation Lending to just the tuples for Book B1
2. *Join* the result of Step 1 to relation Student over Stud_No
3. *Project* the result of Step 2 over Stud_Name

Total tuple I/O: 10,100

In this approach, the sequence of the operations has been changed to the following:

Select-Join-Project

1. *Select* the relation Lending for just the tuples for Book B1.

- This results in reading 10,000 tuples of Lending relation, but only generates a relation with 50 tuples, which will be kept in memory (see assumption).

The number of tuple I/Os: $10,000 + 0 = 10,000$

2. *Join* the result of Step 1 with relation Student over Stud_No.

- This results in reading 100 tuples of Student relation, and joining them with the relation obtained in step 1 which is already in memory. This join produces a relation of 50 tuples, which again will be kept in memory.

The number of tuple I/Os: $100 + 0 = 100$

3. *Project* the result of Step 2 over Stud_Name.

- Same as step 3 of Query Plan A.


Therefore, the total number of tuple I/Os for query plan B is (10,000 + 100)

Total tuple I/O: 10,100

3.3 Comparison of A and B

Ratio (Plan A to Plan B): $1,020,000 / 10,100$ (102 / 1)

V8. 8



Comparison A vs B

- Ratio (Plan A to Plan B):
1,020,000 / 10,100 (102 / 1)
- Tuple I/Os can be further reduced by using indexes


Although plan A and plan B will produce the same end result for the given query, the order in which the required relational operations are executed is different. It is obvious that plan B requires far fewer tuple I/O accesses than plan A, and therefore is far more efficient in terms of performance.

If, furthermore, use was made in Plan B of an index on Book-No for the Lending relation, the number of tuples read in Step 1 would be further reduced from 10,000 to just 50. Similarly, an index on Student.Stud-No would reduce retrievals in Step 2 to only 50 tuples.

One or more similar examples should be shown to the students to demonstrate how tuple I/Os should be calculated and used as a parameter in the comparison of different query execution plans.

4 Four Stages of Optimisation Process

V8. 9

(20 minutes)


Four Stages of Optimisation - 1

1. Convert query into internal form suitable for machine manipulation - e.g. Query tree, Relational algebra
2. Further convert internal form into equivalent and more efficient canonical form, using well-defined transformation rules (laws)

Having considered the above example, now we describe the query optimisation in a more systematic way.

The whole optimisation process generally involves 4 stages (Date):

1. Convert the query into a more suitable internal form.

First, the original query is converted into some internal representation which is more suitable for machine manipulation.

The typical forms used for this representation are:

- query tree;
- relational algebra.

2. Convert to a more efficient canonical form.

This internal representation is further converted into some equivalent canonical form which is more efficient, making use of well-defined transformation rules (see Visuals V8.11 and V8.12).

V8.10

NCC Four Stages of Optimisation - 2

3. Choose set of candidate low-level procedures, using statistics about the database
 - Low-level operation
 - Implementation procedure
 - Cost formula
4. Generate query plans and choose the best (cheapest) plan by evaluating the cost formulae

3. Choose low-level procedures.

Having completed steps 1 and 2, the query optimiser must now decide how to evaluate the transformed query.

The basic principle here is to consider the transformed query representation as specifying a series of low-level operations, using statistical information about the database.

The main results produced in this stage are:

- a set of low-level operations (for example, join, restriction, *etc.*);
- a set of implementation procedures (one for each low-level operation);
- a set of cost formulae (one for each procedure).

4. Generate query plans and choose the best.

In this stage, a set of candidate query plans are generated by combining together the set of candidate implementation procedures. A final decision is made to choose the best (cheapest) of those plans by evaluating the cost formulae.

5 Transformation Rules

V8.11

NCC Transformation Rules - 1

Rule 1
 (A where Restrict-1) where Restrict-2
 ■ A (where Restrict-1 AND Restrict-2)

Rule 2
 A ((Project1) where Restrict
 ■ (A where Restrict) [Project]

Rule 3
 (A [Project-1]) [Project-2] ■ A [Project-2]

Rule 4
 (A Join B) where Restrict-on-A AND Restrict-on-B
 ■ (A where Restrict-on-A) Join (B where Restrict-on-B)

(35 minutes)

Issue Handout 8.2 which details the rules given below.

- *Rule 1*
Transform a sequence of restrictions against a given relation into a single (ANDed) restriction.

$$\begin{aligned} & (A \text{ where Restrict-1}) \text{ where Restrict-2} \\ \equiv & A \text{ (where Restrict-1 AND Restrict-2)} \end{aligned}$$

- *Rule 2*
Transform a restriction of a projection into a projection of a restriction.

$$\begin{aligned} & A ([Project]) \text{ where Restrict} \\ \equiv & (A \text{ where Restrict}) [Project] \end{aligned}$$

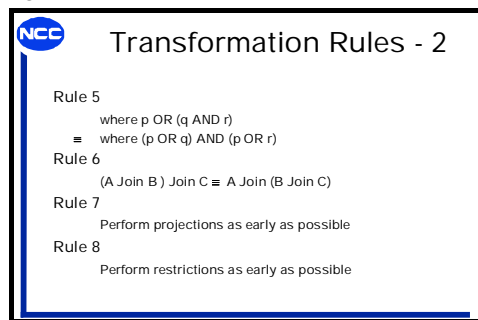
- *Rule 3*
Transform a sequence of projections against a given relation into a single (the last) projection.

$$(A [Project-1]) [Project-2] \equiv A [Project-2]$$

- *Rule 4*
Distributivity (for restrictions and projections).

$$\begin{aligned} & (A \text{ Join } B) \text{ where Restrict-on-A AND Restrict-on-B} \\ \equiv & (A \text{ where Restrict-on-A}) \text{ Join } (B \text{ where Restrict-on-B}) \end{aligned}$$

V8.12



- *Rule 5*
Distributivity (for logical expressions).

$$\begin{aligned} & \text{where } p \text{ OR } (q \text{ AND } r) \\ \equiv & \text{where } (p \text{ OR } q) \text{ AND } (p \text{ OR } r) \end{aligned}$$

- *Rule 6*
Choose an optimal ordering of the joins to keep the intermediate results low in size.

$$(A \text{ Join } B) \text{ Join } C \equiv A \text{ Join } (B \text{ Join } C)$$

- *Rule 7*
Perform projections as early as possible.
- *Rule 8*
Perform restrictions as early as possible.

5.1 Some General Principles

Some general guidelines can be drawn from the above rules. These include:

- Avoid having to build Cartesian products.
- Perform selects and projects before joins if possible to reduce the number of rows which have to be joined.
- Use indexes for selects and joins where appropriate.

At this point, various examples should be shown to the students to illustrate the use of the above rules

6 Database Statistics

(10 minutes)

V8. 13

Database Statistics

System catalogue, or data dictionary
Typical statistics maintained by it include:

- For each base table
 - cardinality
 - number of pages for this table
- For each column of each base table
 - maximum, minimum, and average value
 - actual values and their frequencies
- For each index
 - whether a 'clustering index'
 - number of leaf pages
 - number of levels

Various decisions which have to be made in the optimisation process are based upon the database statistics stored in the system, often in the form of a system catalogue or a data dictionary.

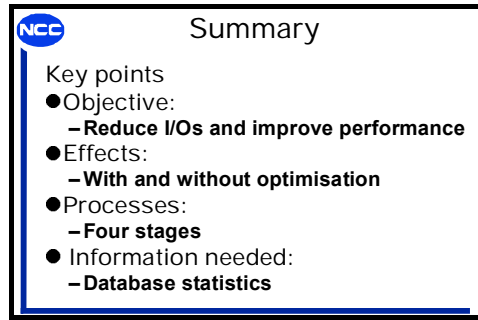
Typical statistics maintained by the system include:

- For each base relation, for example:
 - cardinality of the relation;
 - number of pages for this relation.
- For each *column* of each base relation, for example:
 - number of distinct values in this column;
 - maximum, minimum, and average value for this column;
 - actual values in this column and their frequencies (a histogram).
- For each *index*, for example:
 - whether this is a *clustering index*;
 - number of leaf pages in this index;
 - number of levels in this index.

7 Summary

(5 minutes)

V8. 14



NCC Summary

Key points

- Objective:
 - Reduce I/Os and improve performance
- Effects:
 - With and without optimisation
- Processes:
 - Four stages
- Information needed:
 - Database statistics

The following key points should be emphasised for this topic:

- *Objective* - The importance of query optimisation in terms of minimising I/O operations, speeding up query execution and improving the efficiency and performance of a relational database system.
- *Effects:*
The dramatic effects of optimisation illustrated by the example. Students should be able to compare different query executing plans by calculating the number of tuple I/Os for each plan, as shown in the example.
- *Processes:*
The main tasks involved in each of the four stages of optimisation.
- *Information needed:*
The critical rôle of database statistics in the form of a system catalogue or data dictionary.

Handout 8.1 - Effects of Optimisation - An Example

Query

In order to appreciate the effects of query optimisation on database performance, we use a simple example as an illustration.

Consider the following 3 tables: Student, Lending & Book. The attributes highlighted are the keys for the relevant relations:

```
Student = (Stud_No, Stud_Name, Gender, Address)
Lending = (Lending_No, Stud_No, Book_No)
Book    = ( Book_No, Title, Author, Edition)
```

Consider the following query:

Retrieve the names of students who have borrowed the book B1.

This query can be expressed in SQL:

```
Select  Distinct Student.Stud_Name
From    Student, Lending
Where   Student.Stud_No = Lending.Stud_No
And     Lending.Book_No = 'B1'
```

We further make the following two assumptions:

- The database contains 100 students and 10,000 lendings, of which only 50 are for book B1.
- It is possible to hold up to 50 tuples in memory, without having to write back to disk.

Query Execution Plan A - No Optimisation

In this first approach, the operations required by the query are performed in the sequence:

Join-Select-Project

We calculate the number of database accesses (tuple I/O operations) occurred in each operation.

1. *Join* relations Student and Lending over Stud_no.
 - For each Student row, every Lending row will be retrieved and tested (reading each of the 100 Student rows 10,000 times);
 - Every Lending row will match with one Student row, so the number of joined rows in this intermediate relation is 10,000. These have to be written back to disk (only 50 tuples can be held in memory - see assumptions).

So, the number of tuple I/Os occurred in this step is:

$$(100 * 10,000) + 10,000 = 1,010,000$$

2. *Select* the result of Step 1 for just the tuples for book B1.
 - This results in reading the 10,000 joined tuples (obtained in step 1) back into memory.
 - Then Select produces a relation containing only 50 tuples, which can be kept in memory (see assumption).

The number of tuple I/Os in this step is:

$$10,000 + 0 = 10,000$$

3. *Project* the result of Step 2 over Stud_Name to get result (50 max).
 - This results in reading a relation of 50 tuples (obtained in step 2) which is already in memory, and producing a final relation of no more than 50 tuples, again in memory.

The number of tuple I/O in this step is:

$$0 + 0 = 0$$

Therefore, the total number of tuple I/Os for query plan A is:

$$(1,010,000 + 10,000).$$

Total tuple I/O: 1,020,000

Query Execution Plan B - With Optimisation

In this approach, the sequence of the operations has been changed to the following:

Select-Join-Project

1. *Select* the relation Lending for just the tuples for Book B1.
 - This results in reading 10,000 tuples of Lending relation, but only generates a relation with 50 tuples, which will be kept in memory (see assumption).

The number of tuple I/Os: $10,000 + 0 = 10,000$

2. *Join* the result of Step 1 with relation Student over Stud_No.
 - This results in reading 100 tuples of Student relation, and joining them with the relation obtained in step 1 which is already in memory. This join produces a relation of 50 tuples, which again will be kept in memory.

The number of tuple I/Os: $100 + 0 = 100$

3. *Project* the result of Step 2 over Stud_Name.
 - Same as step 3 of Query Plan A.

Therefore, the total number of tuple I/Os for query plan B is $(10,000 + 100)$

Total tuple I/O: $10,100$

Handout 8.2 - Transformation Rules

- *Rule 1*
Transform a sequence of restrictions against a given relation into a single (ANDed) restriction.

$$\begin{aligned} & (A \text{ where Restrict-1}) \text{ where Restrict-2} \\ \equiv & A \text{ (where Restrict-1 AND Restrict-2)} \end{aligned}$$

- *Rule 2*
Transform a restriction of a projection into a projection of a restriction.

$$\begin{aligned} & A ([Project]) \text{ where Restrict} \\ \equiv & (A \text{ where Restrict}) [Project] \end{aligned}$$

- *Rule 3*
Transform a sequence of projections against a given relation into a single (the last) projection.

$$(A [Project-1]) [project-2] \equiv A [Project-2]$$

- *Rule 4*
Distributivity (for restrictions and projections).

$$\begin{aligned} & (A \text{ Join } B) \text{ where Restrict-on-A AND Restrict-on-B} \\ \equiv & (A \text{ where Restrict-on-A}) \text{ Join } (B \text{ where Restrict-on-B}) \end{aligned}$$

- *Rule 5*
Distributivity (for logical expressions).

$$\begin{aligned} & \text{where } p \text{ OR } (q \text{ AND } r) \\ \equiv & \text{where } (p \text{ OR } q) \text{ AND } (p \text{ OR } r) \end{aligned}$$

- *Rule 6*
Choose an optimal ordering of the joins to keep the intermediate results low in size.

$$(A \text{ Join } B) \text{ Join } C \equiv A \text{ Join } (B \text{ Join } C)$$

- *Rule 7*
Perform projections as early as possible.

- *Rule 8*
Perform restrictions as early as possible.