THE UNIVERSITY OF MELBOURNE

COMP90050
Advanced Database Systems

Winter Semester, 2023

Lecturer: Farhana Choudhury (PhD)

Week 2 part 1

Core Concepts of Database management system

Database performance metrics

- Efficiency/speed
- Effectiveness
- Security & Reliability

Efficiency

- Hardware
  - Disks and I/O bandwidth
  - Main memory
  - Type of architecture
- Software/ DB tuning
  - Types of DB
  - Indexing
  - Query optimisation

Effectiveness

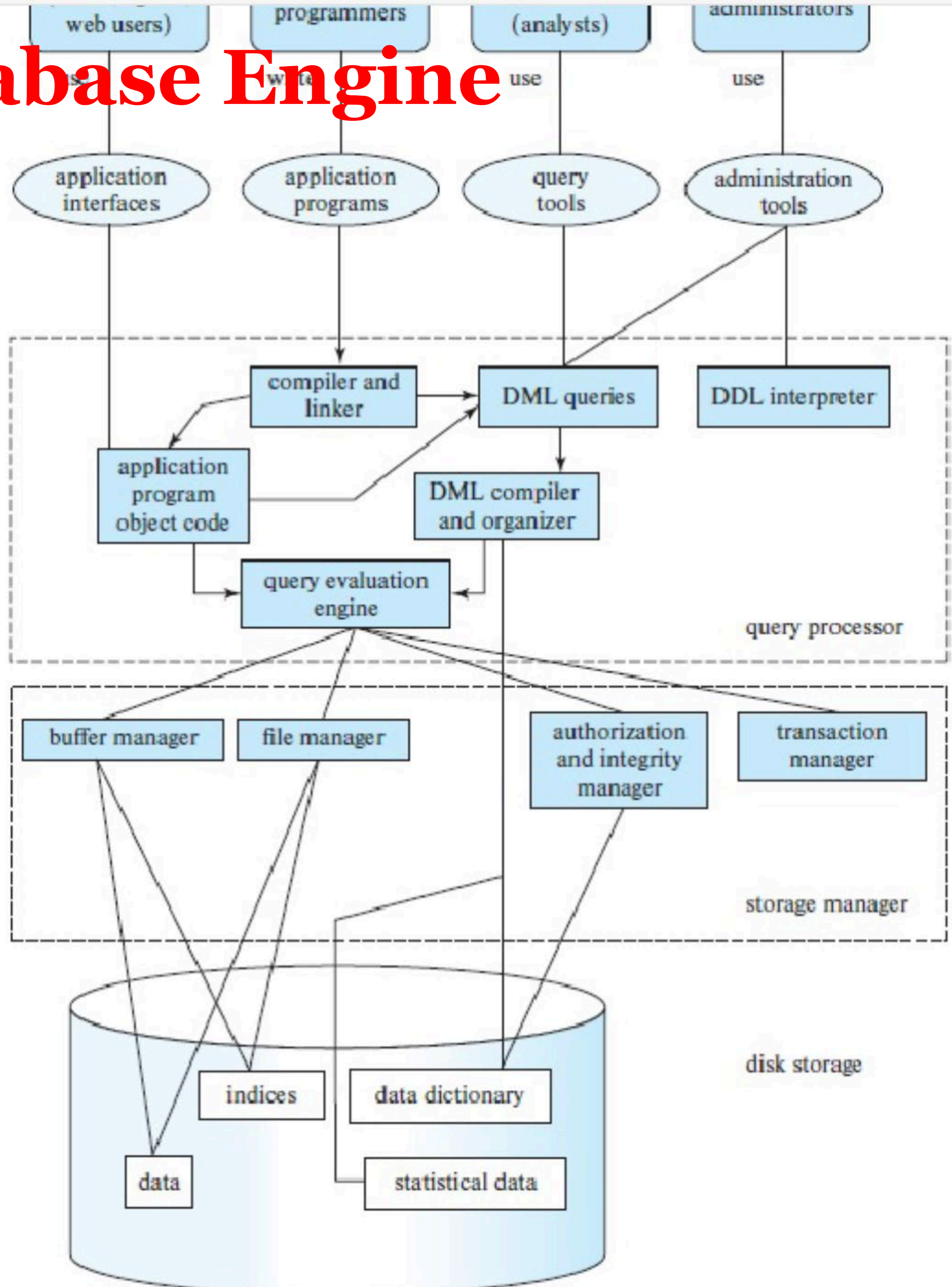
- Concurrent users
- Transactions

Reliability

- Crash recovery
- Fault tolerance
- Data duplication

Hardware:
- Arrangement of multiple disks
- Voting among multiple disks/modules
- Disk block write

Database Engine



Different types of queries from different types of users

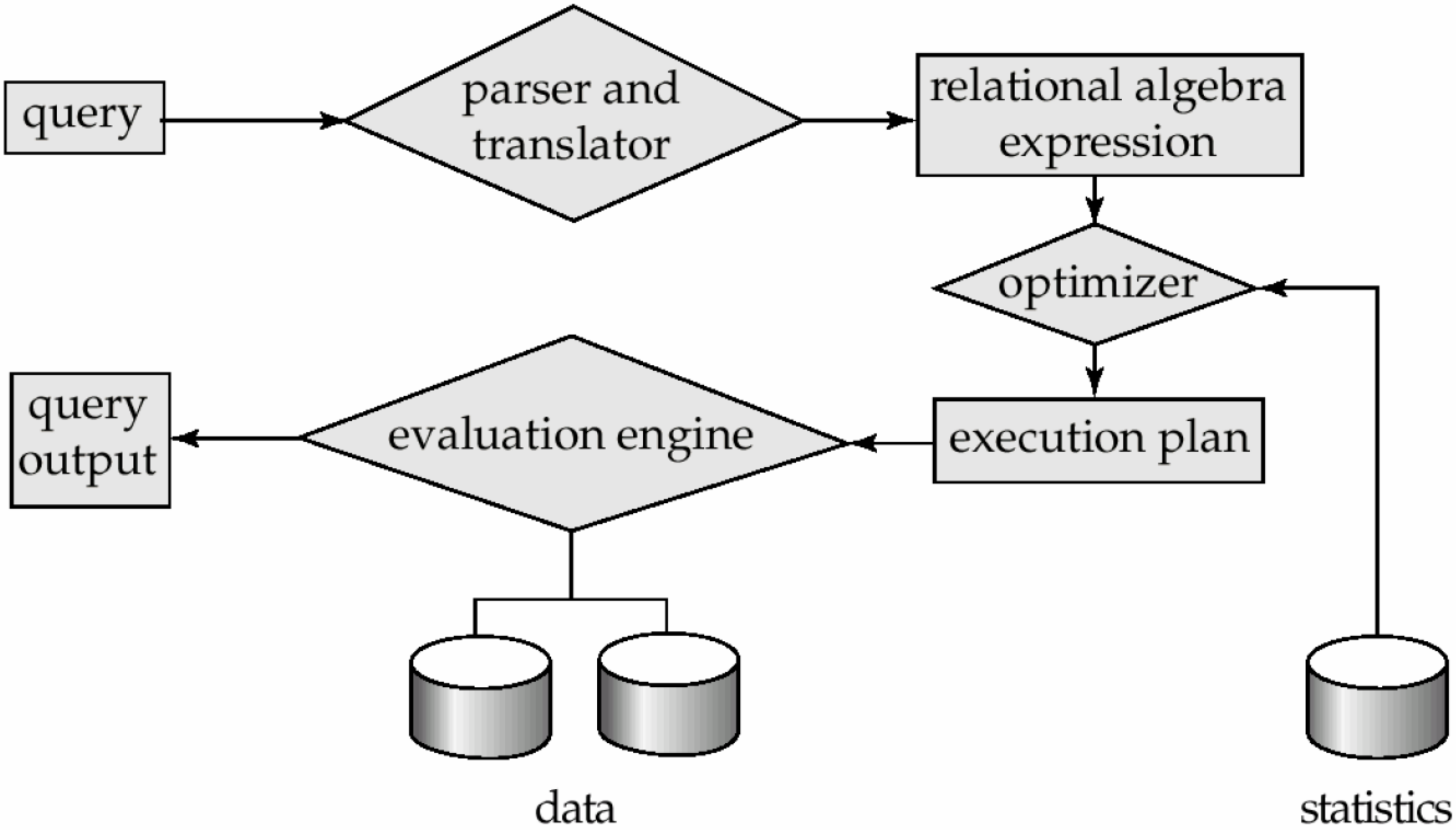
Query evaluation engine

query processor

The storage manager provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.

The storage manager implements several data structures as part of the physical system implementation: (a) data files (the database itself); (b) Indices (to provide fast access to data items; (c) data dictionary (metadata)

Query Processing Steps



Query Processing

A sample SQL query

Select salary

From Employees

Where salary < 60000

- Translate to relational algebra expression
- Make execution plan(s)
- Choose the plan with the least cost (fastest plan)

$$\Pi_{salary}$$

$$\sigma_{salary<60000}$$ (use an index, if any)

Employee

Recall: relational algebra expressions

Select A1, A2, …, An
From r1, r2, …, rn
Where P

Select salary
From Employees
Where salary < 60000

Is same as the following in relational algebra expression:

$$\prod_{A_1, A_2, \ldots, A_n} (\sigma_P (r_1 \times r_2 \times \ldots \times r_m))$$

$$\Pi_{salary}(\sigma_{salary<60000} \text{ (Employees))}$$

Can also be written as:

$$\sigma_{salary<60000} \text{ (}\Pi_{salary} \text{ (Employees))}$$

Recall: join operations

Select *
From r1
Inner Join r2
on T1.a = T2.b

```
SELECT salary
FROM Employees
INNER JOIN Managers
ON Employees.EmpID = Managers.EmpID;
```

…. Is same as the following in relational algebra expression:

$$\Pi_{salary}(\sigma_{Employees.EMPID=Managers.EMPID}\ (\text{Employees X Managers}))$$

More on Joins

$r \bowtie_{\theta} s$

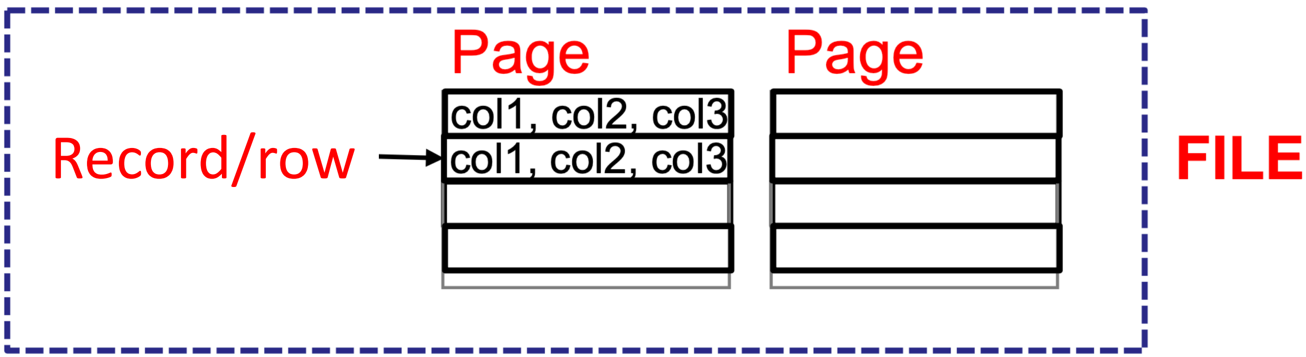
$r \bowtie s$

Natural join

- Here, *r* and *s* are tables
- Theta (θ) is the condition (for example, <,>,=)
- Natural join – joining based on common columns
- Joins are very common and also very expensive

How data are stored

- **Files –** A database is mapped into different files. A file is a sequence of records.
- **Data blocks –** Each file is mapped into fixed length storage units, called data blocks (also called logical blocks, or pages)



How data are stored

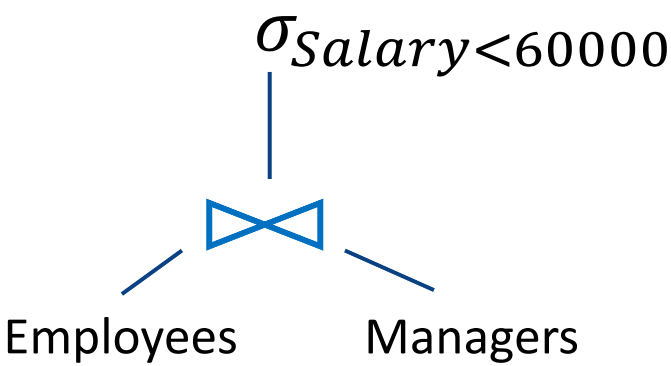
- **Files –** A database is mapped into different files. A file is a sequence of records.
- **Data blocks –** Each file is mapped into fixed length storage units, called data blocks (also called logical blocks, or pages)
- **Cost of a query** – The number of pages/ disk blocks that are accessed from disk to answer the query

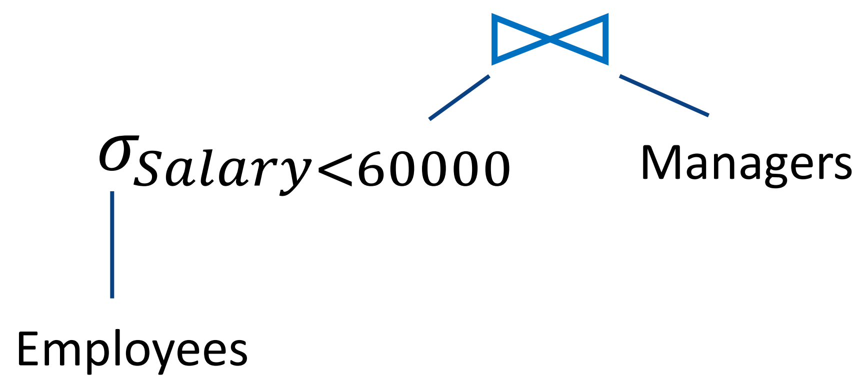
Most dominant cost
(Recall from memory hierarchy!

Query plans and optimisation

Steps in **cost-based query optimisation**

1. Generate logically equivalent expressions of the query
2. Annotate resultant expressions to get alternative query plans
   - Heap scan/Index scan?
   - What type of join algorithm?





Query plans and optimisation

Steps in **<u>cost-based query optimisation</u>**

1. Generate logically equivalent expressions of the query
2. Annotate resultant expressions to get alternative query plans
    - Heap scan/Index scan?
    - What type of join algorithm?

3. Choose the cheapest plan based on estimated cost

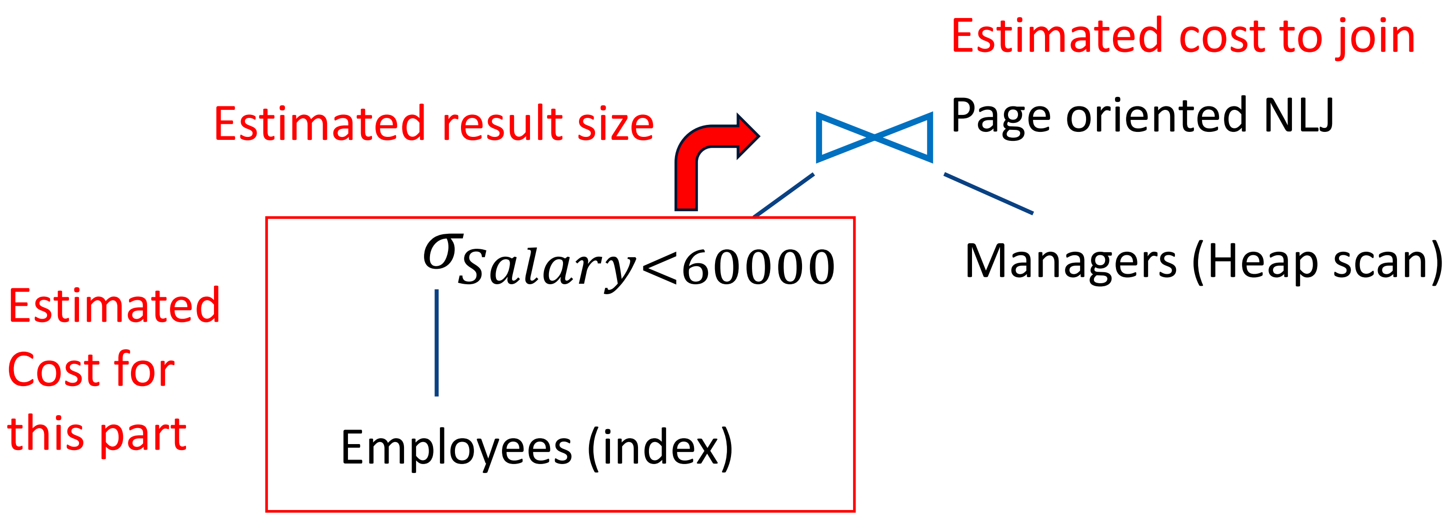
Estimation of plan cost based on:

- Statistical information about tables.

  Example: number of distinct values for an attribute

- Statistics estimation for intermediate results to compute cost
- Cost formulae for algorithms, computed using statistics again

Estimation of query plan cost



How to estimate costs

Step 1: Result size calculation using **Reduction Factor**

$$\sigma_{Salary<60000}$$

Employees

What can go wrong?

Depends on the type of the predicate:

1. Col = value: **RF = 1/Number of unique values (Col)**
2. Col > value: **RF = (High(Col) – value) / (High(Col) – Low(Col))**
3. Col < value: **RF = (val – Low(Col)) / (High(Col) – Low(Col))**
4. Col_A = Col_B (for joins):

RF = 1/ (Max number of unique values in Col_A,Col_B)

How to estimate costs

Step 2: Different options for retrieving data and calculating cost (again, **estimation)**

$\sigma_{Salary<60000}$

Employees (Heap scan)

$\sigma_{Salary<60000}$

Employees (Index scan)

$\sigma_{Salary<60000}$

$\bowtie$

Employees Managers

What can go wrong?

Joins Continued

- Several different algorithms to implement joins exist that the optimizer can look at and pick the best
  - Nested-loop join
  - Page-oriented nested-loop join
  - Merge-join
  - Hash-join
  - …

Joins Contd.

- There is a choice on how to run a query on the server

- Choice is based on cost estimates:
  - statistics,
  - table sizes,
  - available indices
  - …

- **Decisions effect performance dramatically**

A Simple Nested-Loop Join

- To compute a theta join

for each tuple t_r **in** r **do begin**
    for each tuple t_s **in** s **do begin**
        test pair (t_r,t_s) to see if they satisfy the join condition theta (θ)
        if they do, add $t_r \bullet t_s$ to the result.
    end
end

A Simple Nested-Loop Join Contd

- ***r*** is called the **outer relation** and ***s*** the **inner relation** of the join.
- Requires no indices and can be used with any kind of join condition.
- **Expensive since it examines every pair of tuples** in the two relations.
- Remember that for every retrieval, especially for a different item from the the disk in a nonconsecutive location we pay a **seek time as a penalty**, this is where it could happen a large number of times.
- Could be cheap if you do it on two small tables where they fit to main memory though (*disk brings the whole tables*).

Let's Calculate the Costs

Let's see an example with the following bank database:

- Number of records of *customer*: 10,000 *depositor*: 5000
- Number of Pages of *customer*: 400 *depositor*: 100

In the worst case, if there is enough memory only to hold one page/block of each table, the estimated cost is

$$\boldsymbol{b_r + (n_r * b_s)}$$ Page access

So Two Options Are

With *depositor* as the outer relation:

$100 + (5000 * 400) = 2{,}000{,}100$ page access,

With *customer* as the outer relation:

$400 + (10000 * 100) = 1{,}000{,}400$ page access

If you had **1000,000 customers, then you would wait several hours for one simple join**!

A Better Way: Page-Oriented Nested-Loop Join

Variant of nested-loop join in which every page of inner relation is paired with every page of outer relation.

```
for each page Br of r do begin
    for each page Bs of s do begin
        for each tuple tr in Br do begin
            for each tuple ts in Bs do begin
                Check if (tr,ts) satisfy the join condition
                if they do, add tr • ts to the result.
            end
        end
    end
end
```

Let’s Calculate the Costs

Let’s see an example with the following bank database:

- Number of records of *customer*: 10,000 *depositor*: 5000
- Number of Pages of *customer*: 400 *depositor*: 100

In the worst case, if there is enough memory only to hold one page/block of each table, the estimated cost is

$$\boldsymbol{b_r + (b_r * b_s)}$$ Page access

So Two Options Are

With *depositor* as the outer relation:

100 + (100 ∗ 400) = 40100 page access,

With *customer* as the outer relation:

400 + (400 ∗ 100) = 40400 page access

Several order or magnitude faster than NLJ!

Recap: Query plans and optimisation

Steps in **cost-based query optimisation**

1. Generate logically equivalent expressions of the query
2. Annotate resultant expressions to get alternative query plans
   - Heap scan/Index scan?
   - What type of join algorithm?
3. Choose the cheapest plan based on estimated cost

Why it’s important to have a good query optimiser?

- It’s the heart of query efficiency
- Generating all equivalent expressions exhaustively - very expensive
- Must consider the interaction of evaluation techniques when choosing evaluation plans. Choosing the cheapest algorithm for each operation independently may not yield best overall cost
- Estimations of the result size may not be accurate

So, In Real Life….

Cost-based optimization is expensive, thus….

- Systems may use heuristics to reduce the number of choices that must be made in a cost-based fashion

- Heuristic optimization transforms the query-tree by using a set of rules that typically (but not in all cases) improve execution performance:

1. Perform selections early (reduces the number of tuples)
2. Perform projections early (reduces the number of attributes)
3. Perform most restrictive selection and join operations (i.e., with smallest result size) before other similar operations

Real Life Contd

- Some systems use only heuristics, others combine heuristics with cost-based optimization

- *Optimizers often use simple heuristics for very cheap queries, and perform exhaustive enumeration for more expensive queries*

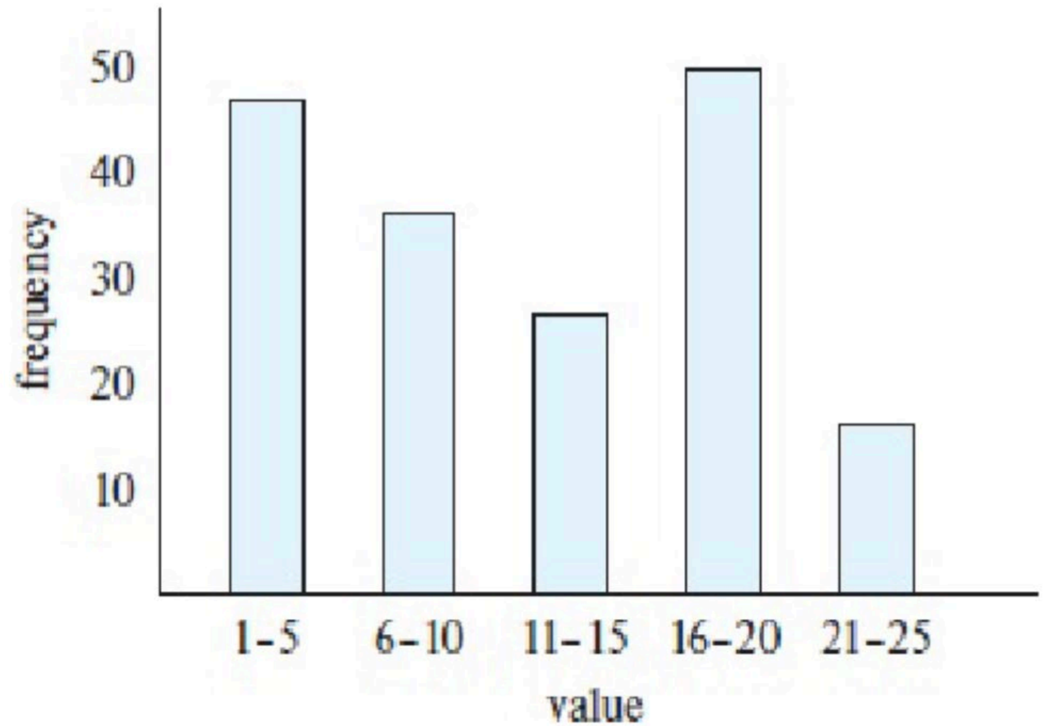
Real Life Contd

- Some systems use only heuristics, others combine heuristics with cost-based optimization

- *Optimizers often use simple heuristics for very cheap queries, and perform exhaustive enumeration for more expensive queries*

Further optimisation: Better estimation of reduction factors

1. Sampling
2. Histograms



Histogram width – learn from data and query

Adaptive plans

Wait for one/some parts of a plan to execute first, then choose the next best alternative

Hash-join if >100 pages from Employees, otherwise PNLJ (just as an example)

$\bowtie$

$\sigma_{Salary<60000}$ Managers

Employees

Readjust statistics: learning from mistake

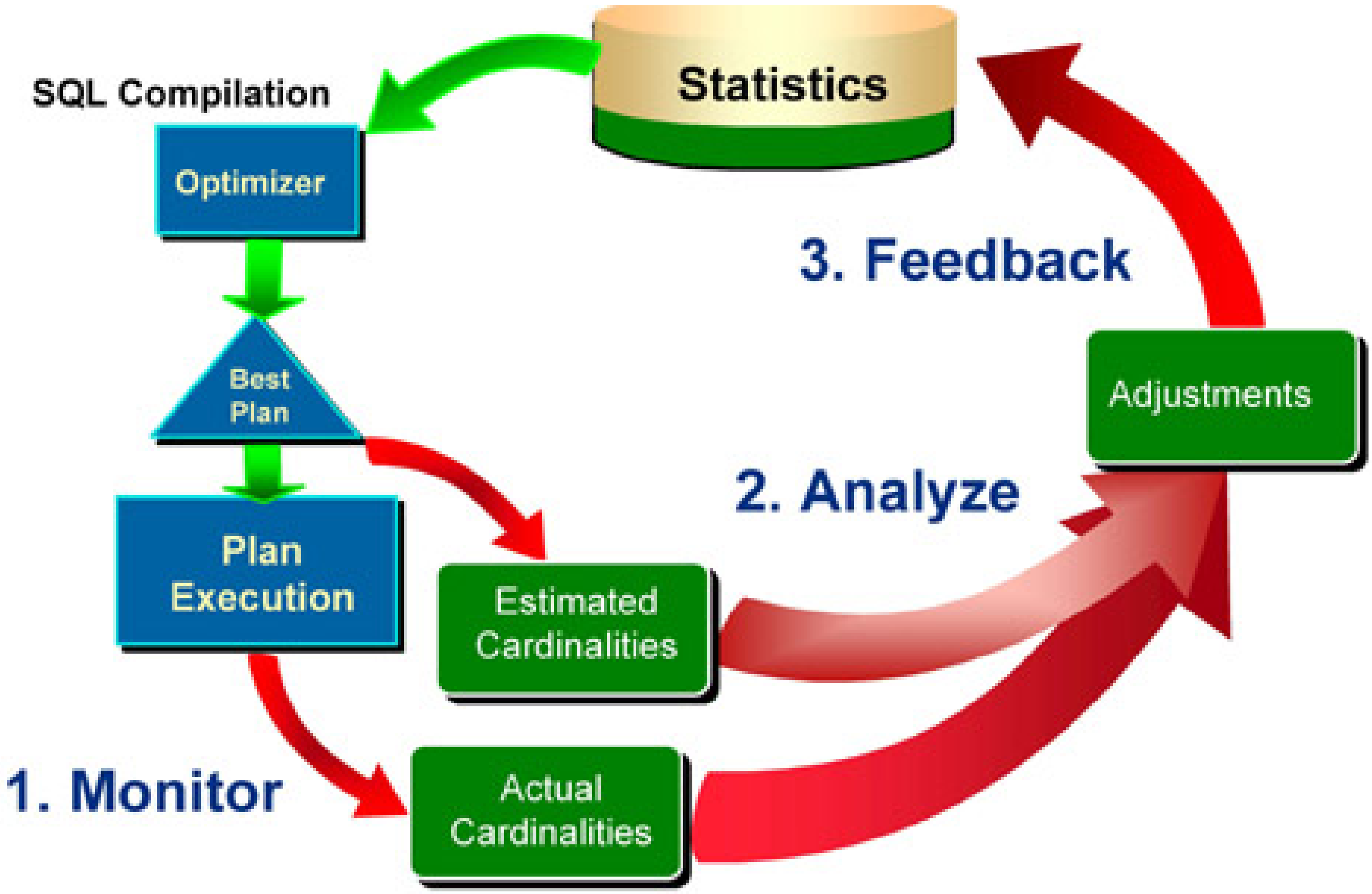
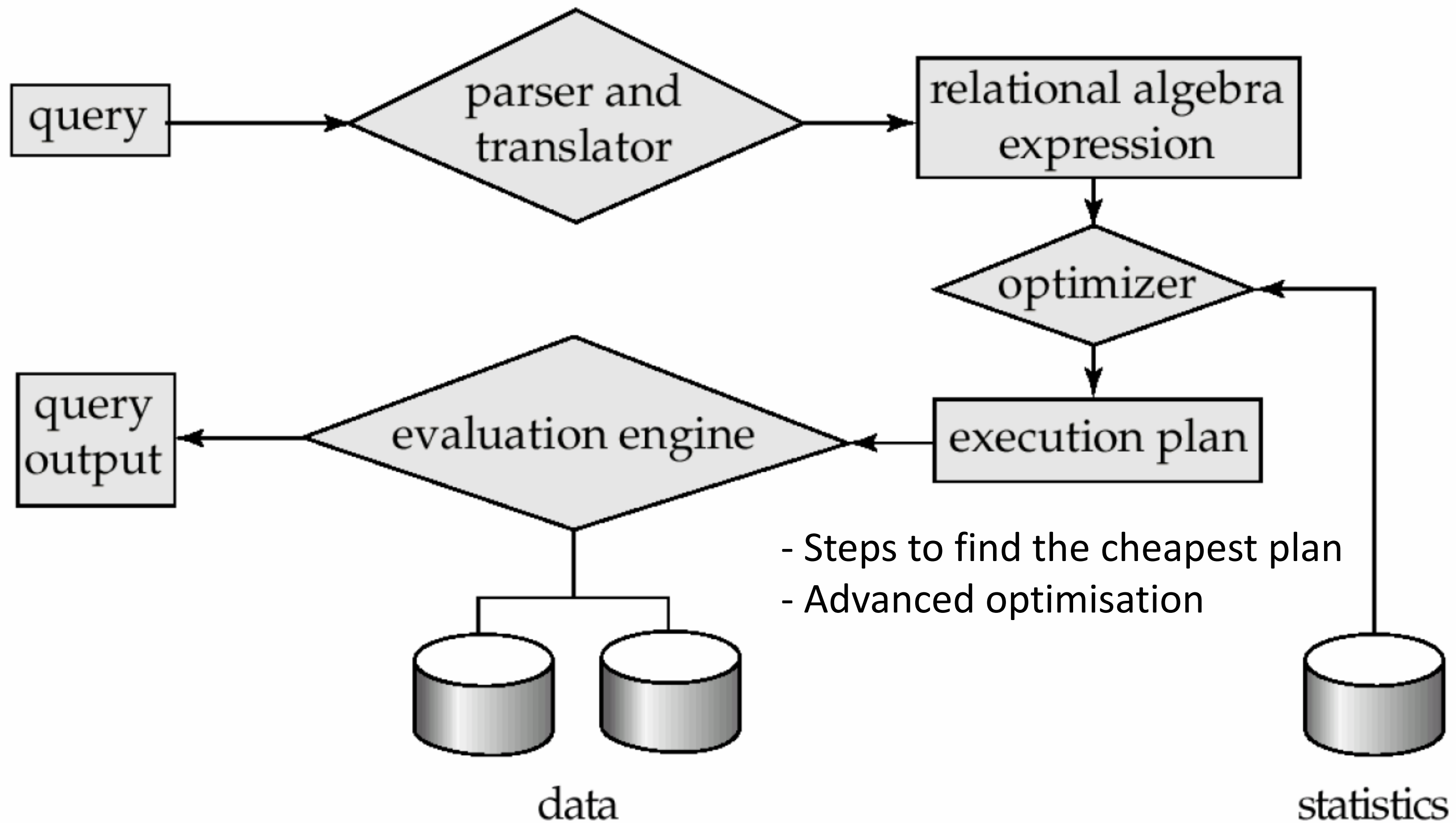


Image courtesy: Allison Griffin

Summary



- Steps to find the cheapest plan
- Advanced optimisation