COMP90050 ADBS

授课老师：Rita

1 – Query Processing & Optimisation

基本概念

- 8 continuous bits = 1 byte
- 4000/ 8000 continuous bytes = 1 block
- Bit – b; byte – B; block – A

record < block < file

How data is stored in disk?

record => 行

- 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)

e.g., size of each record: 55 byte; fixed size of 1 data block: 4096 byte

$\frac{4096}{55} \simeq 74.47$ => 74 条记录

一条数据不能被拆到两个block中

Table – "**Employee**"

ID	Name	Age
1001	A	25
1002	B	32
1003	C	19
1004	D	27
1005	E	40
1006	F	36

Table – "**Manager**"

ID	Department
1001	a
1003	b
1004	c
1006	d

Table – "**Location**"

ID	Country	City
1001	China	Beijing
1002	Australia	Melbourne
1004	China	Shanghai
1006	Australia	Sydney

- Attribute (column) 3
- Record (row)
- Query
  - Please tell me the name of all employees who is over 25 filter
  - please tell me the name of all managers → join on ID →
  - Please tell me the name of all managers located in Australia
- Join

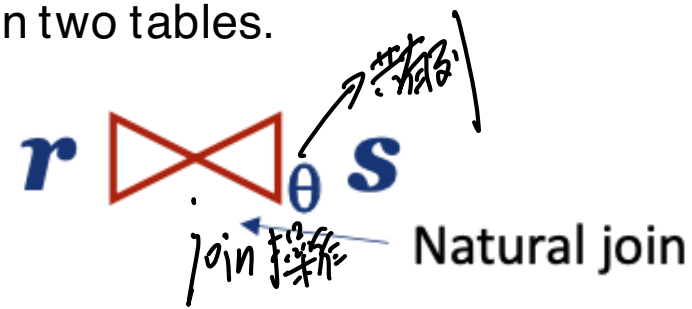
```
Select Name
from (Employee inner join Manager)
On Employee.ID = Manager.ID;

Select Name
from (A inner join location)
On A.ID = Location.ID
Where Location.country == 'Australia';
```


Join

Join is a very common and very expensive operation

Different types of join: inner join, outer join… (we focus on inner join)

Natural join: a join operation that can be performed of a column that is common in two tables.



r: outer relation
s: inner relation
r and s are two tables
Theta is the common column.

Relational Algebra Expressions

Select * *from* T1

inner join T2

on T1.a = T2.b

all attributes

```
SELECT *
FROM Employees
INNER JOIN Managers
ON Employees.ID = Managers.ID;
```

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

$$\Pi_{*}(\sigma_{Employees.ID=Managers.ID}(\text{Employees } \underline{X} \text{ Managers}))$$

all attribute

Common attribute

join

Relational Algebra Expressions

select A1, A2, ..., An
from r1, r2, ..., rm
where P

Select salary
From Employees
Where salary < 60000

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

由里向外拆分.

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

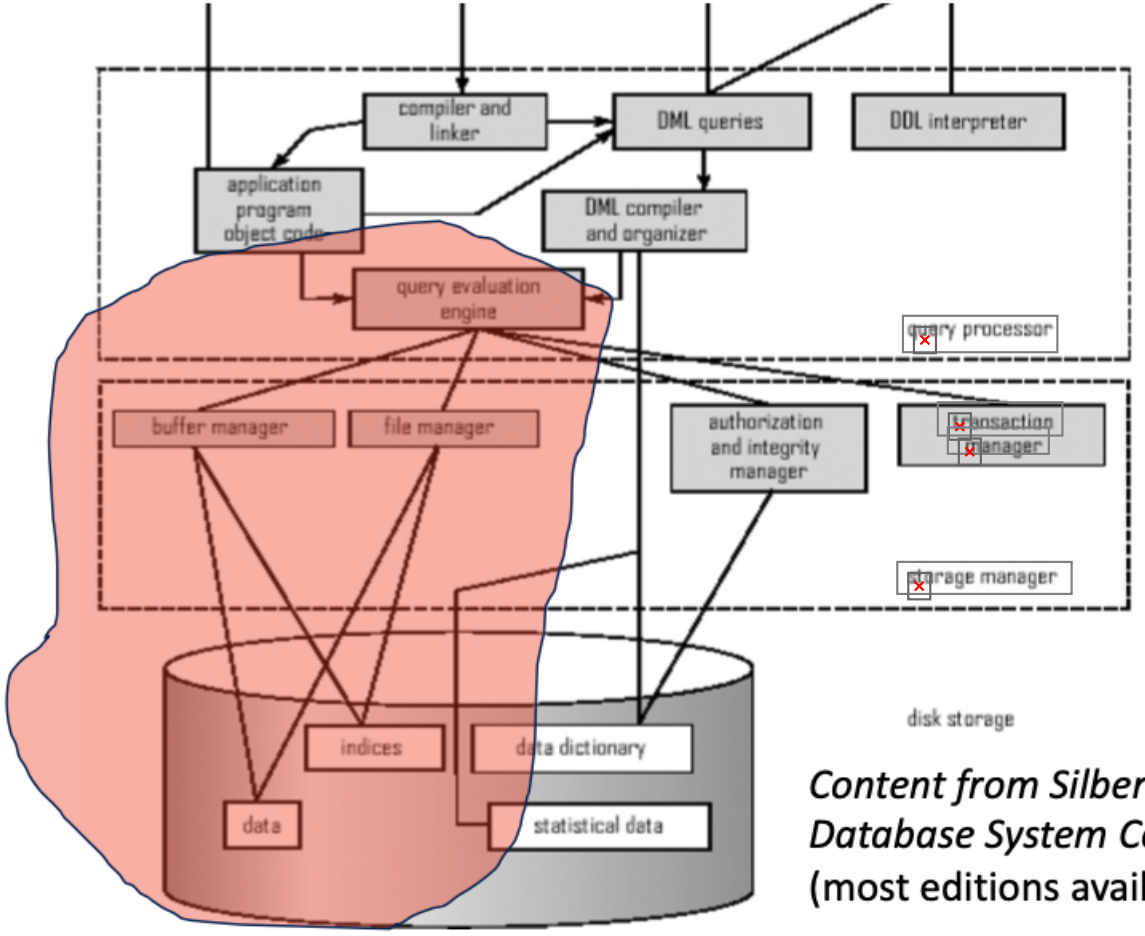
Equivalent Expressions

Can also be written as:

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

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

DB Engine



Content from Silberschatz et al Database System Concepts
(most editions available online)

- **Hardware**
  - – The speed of the processor
  - – Number of processors
  - – Number of disk drives and I/ O bandwidth
  - – Size of main memory
  - – Communication network
  - – Type of architecture

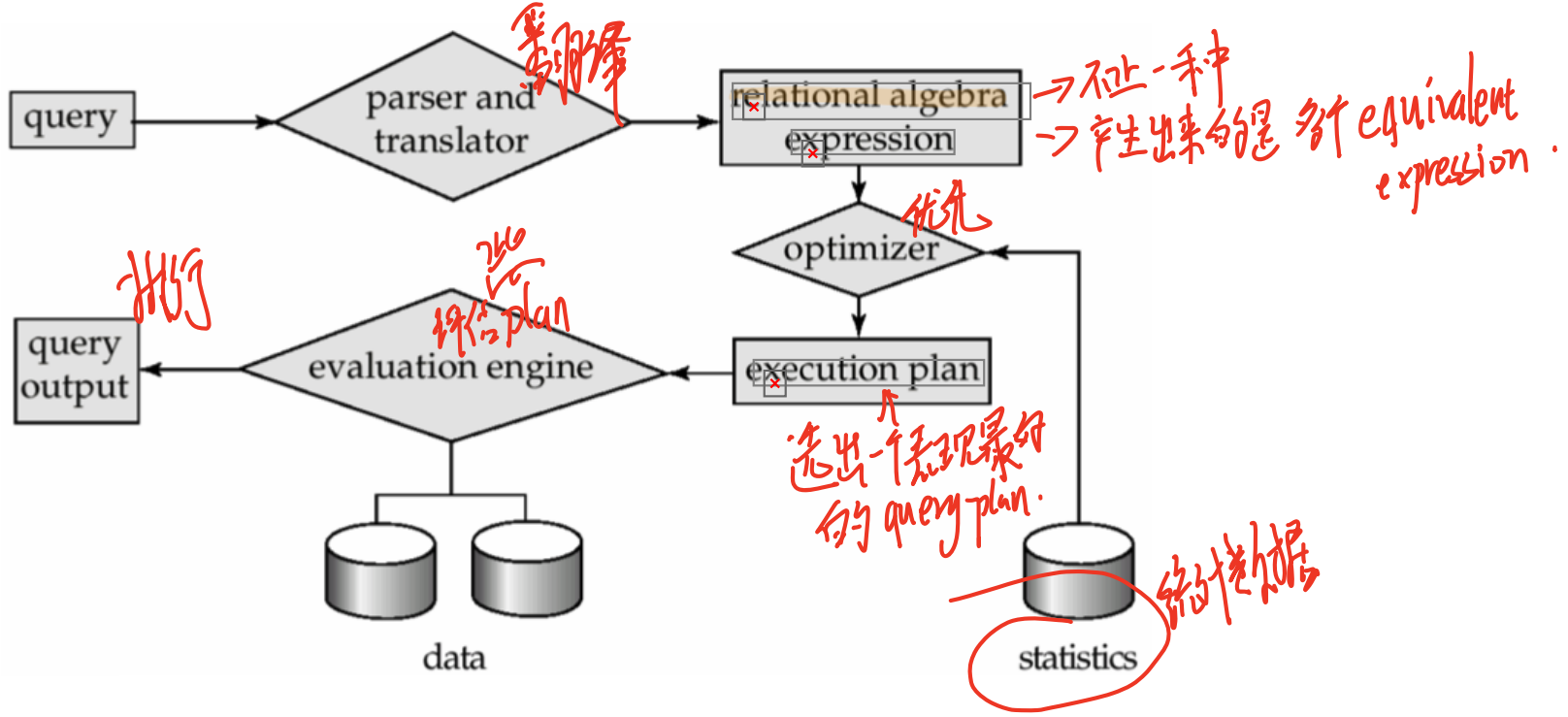
DB Engine

- **Software**
  - – Type of database technology used for a given application
- **Database tuning, crash recovery**
  - – Indexing parameters
  - – Data duplication
  - – Sharing data, etc

* 一个query会有多个plan产生
* 优化后会选取一个最优的plan.

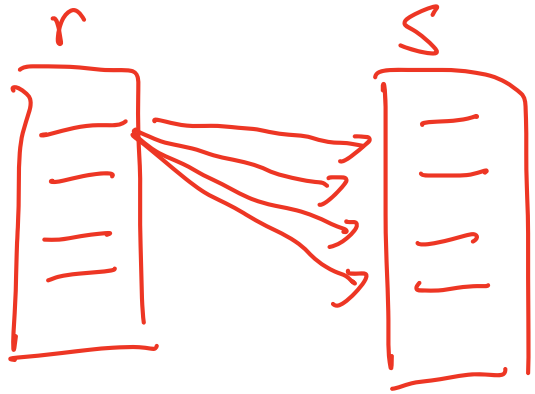
Query Processing Steps

E.g., select Salary From Employees Where Salary < 60000



- Translate query to relational algebra expression
- Make execution plan

r ⋈θ s
outer inner
condition



Nested-loop Join (不用index)

record

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

- Requires no indices because it checks everything in r against everything in s (既然不用index更快速查找)
- Expensive since it examines every pair of tuples in the two relations. (record多的话)
- Could be cheap if you do it on two small tables where they fit to main memory (disk brings the whole tables with first block access). (两个表会容易)

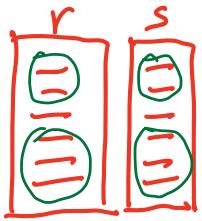
definition of cost: how many blocks are read from disk?

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

n : record
b : block

accesstime: $n_r * b_s + b_r$ block transfers, and

$n_r + b_r$ seeks



Block Nested-loop Join (Page-Oriented Nested-loop Join)

```
for each block B_r of r do begin
    for each block B_s of s do begin
        for each tuple t_r in B_r do begin
            for each tuple t_s in B_s do begin
                Check if (t_r, t_s) satisfy the join condition
                if they do, add t_r • t_s to the result.
            end
        end
    end
end
```

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

把nested loop join 中 n_r 换成 b_r 即可

$$b_r * b_s + b_r \text{ block transfers} + 2 * b_r \text{ seeks}$$

n_r - >b_r

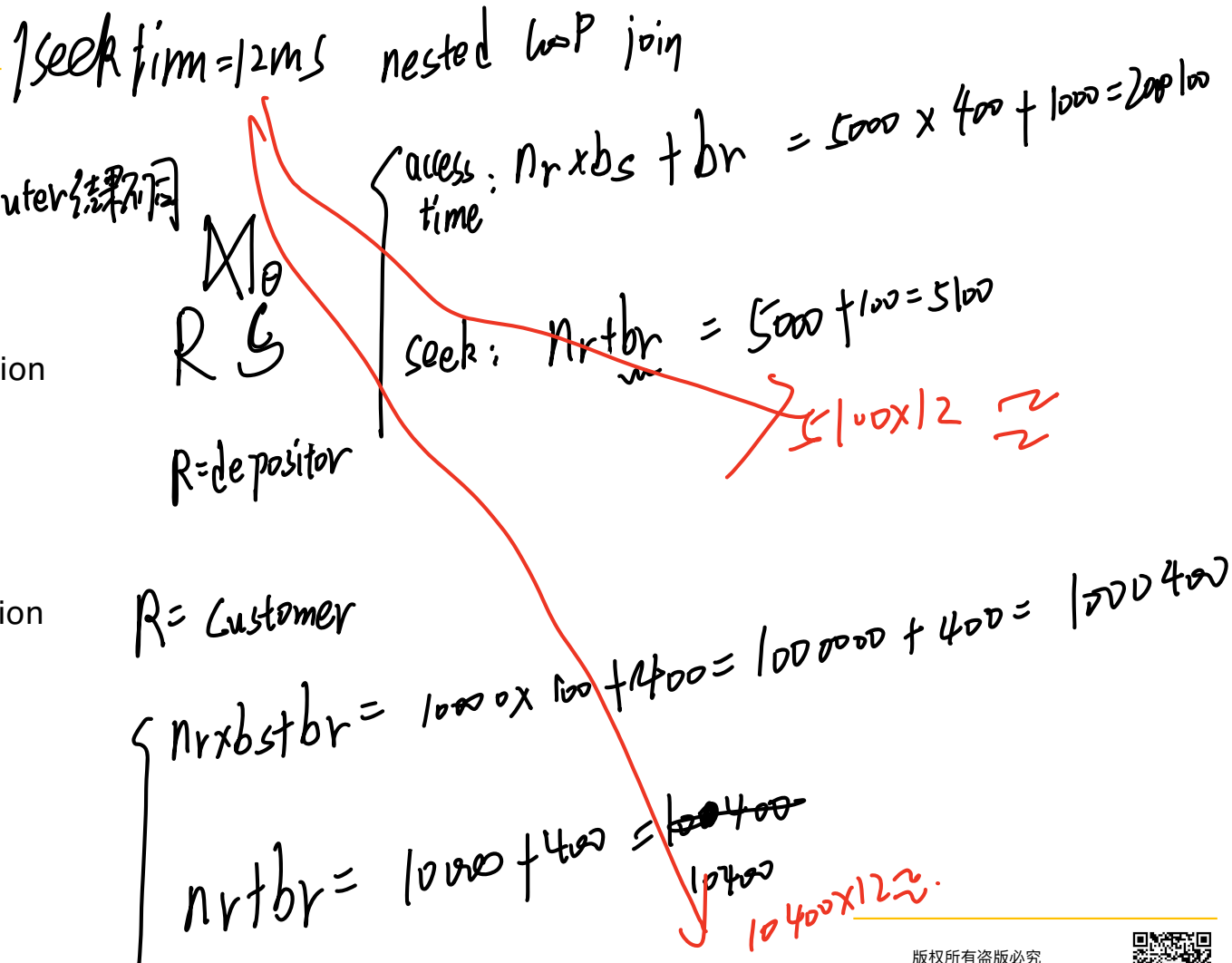
e.g.,

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

Nested-loop Join

Depositor as the outer relation

Customer as the outer relation



block transfer: $b_r \times b_s t_{br}$

access time: $2 \times b_r$

Block Nested-loop Join (Page-Oriented Nested-loop Join)

Depositor as the outer relation

M RS

$b_{depositor} \times b_{customer} + b_{depositor}$

$= 100 \times 400 + 100$

seek: 2×100

Customer as the outer relation

$400 \times 100 + 400$

2×400

Customer做为 outer relation 时查询更快(推导错误)

Query Optimization

Query optimization is about the right choices and annotations

What to optimize? join的顺序优化和物理属性的变化

- e.g., I have 3 tables, which two can I join first?
- Join algorithms

Query Plans

Alternatives 备选

Execution Plan (最终执行策)

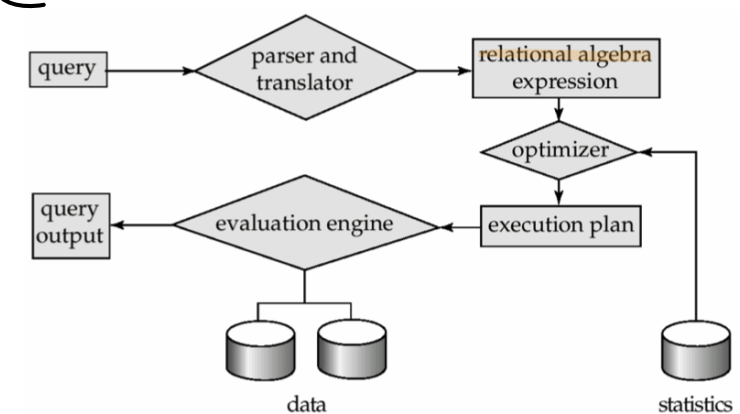
Final decision

How do we make the choices?

steps in cost-based query optimization

Cost: how many blocks are read from disk

1. Generate logically equivalent expressions of the query
2. Annotate resultant expressions to get alternative query plans
3. Choose the cheapest plan based on estimated cost



选择plan依据

Estimation of plan cost based on:

(1) - Statistical information about tables. Example: ⇒

number of distinct values for an attribute

ID	Name	Age
1	A	18
2	B	21
3	C	19
4	D	20

18,21,20

↓

distinct.

- Statistics estimation for intermediate results to compute cost of complex expressions

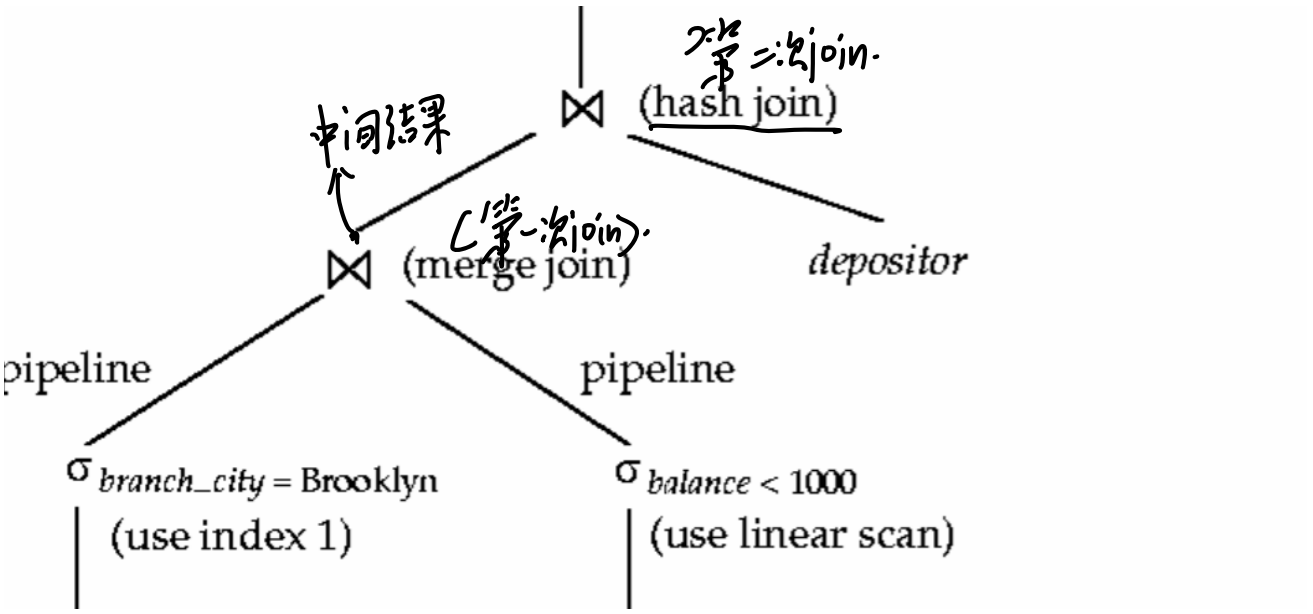
A+B+C (A⋈B 中间结果)

- Cost formulae for algorithms, computed using statistics again

算法cost

2 – Query Optimization in Real Life

Query Optimization is about the right choice on a graph



How to generate alternatives?

- Query optimizers use equivalence rules to systematically generate expressions equivalent to the given expression
- One can generate all equivalent expressions exhaustively (所有 plan 都生成)
- The above approach is very **expensive** in space and time though.

(In query optimizer, some expressions are not generated if they are for sure very complex)

若有的真的可能

But

- Must consider the interaction of evaluation techniques when choosing evaluation plans
- Choosing the cheapest algorithm for each operation independently may not yield best overall algorithm (每步最好的，不一定overall最好)

e.g., merge- join may be costlier than hash- join, but may provide a sorted output which could be useful later (the sorted result may be benefit to the later operation)

In Real Life

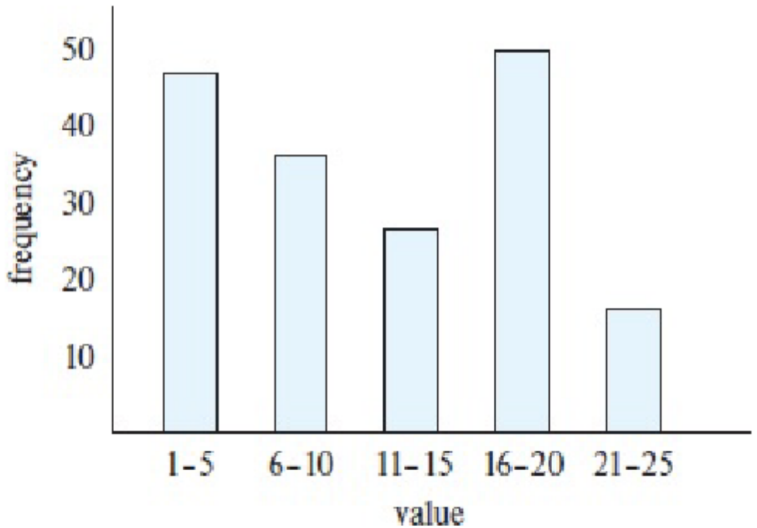
- Practical query optimizers incorporate elements of the following two broad approaches:
  1. Search all the plans and choose the best plan in a cost- based fashion.
  2. Uses **heuristics** to choose a plan.
- Systems may use heuristics to **reduce the number of choices** that must be made in a cost- based fashion (because cost- based optimization is expensive)

- 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

- 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 Optimisations
  - Sampling
  - Histogram



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

3 – Query Cost in Practice

Troubleshooting to manage costs (e.g., Query Store in Microsoft)

监控 query

(Query store: SQL server management studio for monitoring)

识别较差的 query 表现

- Identify 'regressed queries' - Pinpoint the queries for which execution metrics have recently regressed (for example, changed to worse).

- Track specific queries - Track the execution of the most important queries in real time. (e.g., most frequently asked queries)

跟踪重要的 query 的执行

When you identify a query with suboptimal performance
(没有达到最好的performance)

对无理数的习惯一个表现好的子集判 force execution

- Force a query plan instead of the plan chosen by the optimizer
- Do we need an **index**? - - - quickly find the data in the query
- Enforce statistic recompilation (重新编译)
- Rewrite query? (with parameters)

Next page

Parameter in query

给一类query plan 假定为同一个参数。

Query rewriting with parameters for execution plan reuse

```
SELECT *
FROM Product
WHERE categoryID = 1;
```

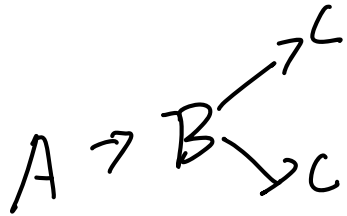
```
SELECT *
FROM Product
WHERE categoryID = 4;
```

They use the same plan!

We expect the optimizer to generate essentially the same plan and reuse the plans - parameterize

```
DECLARE @MyIntParm INT
SET @MyIntParm = 1
EXEC sp_executesql
  N'SELECT -
  FROM Product
  WHERE categoryID = @Parm',
  N'@Parm INT',
  @MyIntParm
```

"1" & "4"



To further lower query cost

- **Store derived data**
  - When you frequently need derived values
  - Original data do not change frequently
- **Use pre-joined tables**
  - When tables need to be joined frequently
  - Regularly check and update pre- joined table for updates in the original table
  - May still return some 'outdated' result (pre- joined tables are not updated)