University of Glasgow | School of Computing Science

Introduction to Data Science and Systems

Zaiqiao Meng
zaiqiao.meng@glasgow.ac.uk

Lecture 11 - Query Optimisation

Query Optimization

- **Heuristic Optimization:**
  - **Task:** Apply a set of *inferential* rules to *transform* a query to an **equivalent** and **efficient** query using Relational Algebra
  - Equivalent query *produces* exactly the same results

- **Cost-based Optimization:**
  - **Task 1:** provides *alternative* execution plans and *estimates* (**predicts**) their costs
  - **Task 2:** chooses the plan with the **minimum cost;**

  - **Objective / Cost Function:** block accesses, storage/memory costs, in-memory computing costs, network bandwidth…

Roadmap

- *Fundamental* component in Cost-based Optimization:
  - **Selectivity**: *fraction* of tuples satisfying a selection condition
- **Challenge 1**: Prediction of selection cardinality, i.e., *predict* the *number* of tuples retrieved given a selection query
- **Challenge 2**: Refinement of expected cost of selection queries w.r.t. selectivity

Cost-based Optimization

- We *need* statistical information to estimate the cost of a query
- DB catalog keeps information per *file/relation* and *attributes*
- **Per Relation File:**
  - *number* of records (*r*)
  - (*average*) *size* of each record (**R**)
  - *number of blocks* (*b*)
  - *blocking factor* (*f*) i.e., records per block
  - *Primary File Organization*: heap, hash, or sequential file over an attribute
  - *Index* or *Indexes*: primary, clustering index, secondary index, B+ Trees over attributes
- **Per *each* Attribute A:**
  - *Index level* (*x*) of attribute *A*
  - Number *of* Distinct Values: NDV(A)
  - Range: MAX{A} and MIN{A}
  - *Selectivity* (*sl*): the *fraction* ($\in[0,1]$) of tuples satisfying a *predicate constraint* on attribute A
  - *Probability Distribution Function* **P(A ≤ *a*)**, if A is a real-valued attribute (or Probability Mass Function for discrete domain)
    - A *good* approximation of *frequency* of values: **histogram**

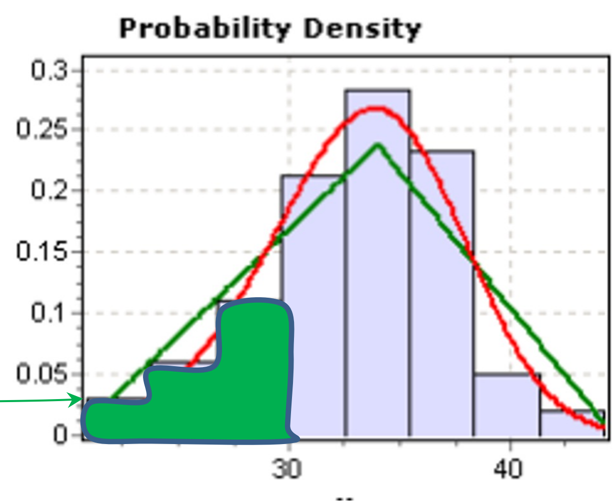
Cost-based Optimization

`SELECT * FROM EMPLOYEE` → Result set cardinality: **1000 tuples**

`SELECT * FROM EMPLOYEE WHERE Salary = 40K` → Result set cardinality: **10 tuples**

Selectivity(Salary) = 10/1000 = 0.01 or **1%** of the tuples satisfy 'Salary=40K'

P(A ≤ 30) = *integral* from 0 to 30 = 0.175 or 17.5%



Selection Cardinality

Challenge: Given a file with *r* records and an *equality* condition over A, *predict the expected number* of records satisfying this condition **without scanning the file**

In other words, *predict*: $\boldsymbol{sl(A) \cdot r}$

- *Selection Cardinality*:

$$s = r \cdot sl(A) \leq r$$

i.e., *average* number of records satisfying an equality condition over A

Fact 1: Selectivity prediction is *indeed* difficult! (sometimes, *intractable*)

Fact 2: Selection cardinality indicates which selection operation should be executed *first* to retrieve as (a) few tuples as possible

Fact 3: Heuristic Optimization, push this selection operation *far down* the query plan tree!

Selectivity Prediction

Solution 1 (no assumption):

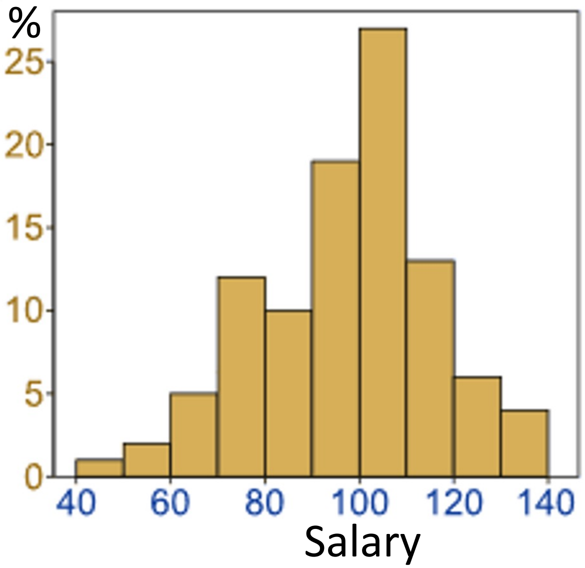
- Estimate the probability density function (distribution of values) or the histogram of the attribute A
- Then: a *good* estimate for *sl*(A = *a*) ≈ P(A = *a*), which depends on the value of *a*

```
SELECT * FROM EMPLOYEE WHERE Salary = 50K
```

P(A=50) = 0.02 = *sl*(A = 50); *s* = 0.02**r***

```
SELECT * FROM EMPLOYEE WHERE Salary = 100K
```

P(A=100) = 0.27 = *sl*(A = 100); *s* = 0.27**r***



Selectivity Prediction

Solution 2 (assumption):

- Accept assumptions on the distribution of the values
- Then: provide a *less representative* prediction for *sl*(A)
- *All* values are *uniformly* distributed, i.e., *sl*(A = *a*) ≈ *constant* independent on *a* value (for all *a* values!)

```
SELECT * FROM EMPLOYEE WHERE Salary = 50K
```

P(A=50) = 0.21 = *sl*(A = 50); *s* = 0.21**r***

```
SELECT * FROM EMPLOYEE WHERE Salary = 100K
```

P(A=100) = 0.21 = *sl*(A = 100); *s* = 0.21**r***

Selectivity Prediction

Fact: Consider an *equality* condition on a **key** attribute A. Then:

$$sl(A) = 1/r$$

since *only one* tuple satisfies the condition; *selection cardinality* = 1 tuple

Example:

```
SELECT * FROM EMPLOYEE WHERE SSN = '12345678'
```

Fact: it returns only *one* tuple since A is SSN.

Assume: r = 1,000 employees, then sl(SSN) = 1/r = 0.001.

Note: the *smaller* the *sl* is, the higher the *desirability* of executing this selection *first*, *i.e., move* this query *far down the tree*; liaise with heuristic optimization

Selectivity Prediction

Fact: Consider an *equality* condition on a ***non-key*** attribute A, which has **NDV(A)** number of distinct values. Then, a *not-so-good* estimate is:

$$sl(A) = (r/\text{NDV}(A))/r \text{ or } \mathbf{sl(A) = 1/NDV(A)}$$

Why?

Because: Under assumption, *all* records are **uniformly distributed** across the *distinct* values, thus, $sl(A = a)$ is the probability of selecting the value of a out of the NDV(A) distinct values

Proof: r/NDV(A) is the number of tuples having a distinct value. Hence, the fraction is: (r/NDV(A))/r = 1/NDV(A)

Note: *Selection cardinality* = r/NDV(A) tuples satisfy an equality condition

Selectivity Prediction

```
SELECT * FROM EMPLOYEE WHERE DNO = 5;
```

Consider: A := DNO; NDV(DNO) = 10 departments, *r* = 1000 employees, and the employees are **evenly distributed** across those departments

Then: distribute 1000/10 = 100 employees per department

sl(DNO) = 1/NDV(DNO) = 0.1 or **10%**

```
SELECT * FROM EMPLOYEE WHERE DNO = 4;
```

Again: sl(DNO) = 1/NDV(DNO) = 0.1 or **10%**

But, in reality, the **probability** of having a *uniform distribution* over an arbitrary attribute is almost **zero** ☹…*thus* we adopt histograms*

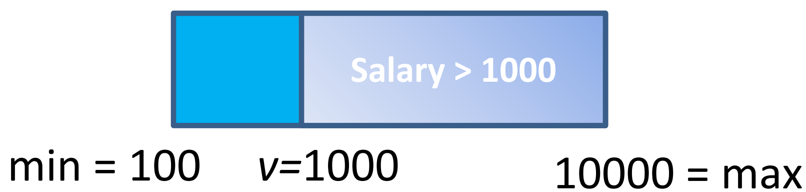
[*] Yannis E. Ioannidis et al; 1996. *Improved histograms for selectivity estimation of range predicates*. ACM SIGMOD'96 NY, USA, 294-305.

Range Selectivity

Fact: Given a *range query* **A ≥ *v*** on A ∈ [min(A), max(A)] and assume uniform value distribution in [min(A), max(A)], then:

$sl(A) = 0$ if $v >$ max(A)

$sl(A) = (\max(A) - v)/(\max(A) - \min(A)) \in [0, 1]$




```
SELECT * FROM EMPLOYEE WHERE Salary ≥ 1000;
```

Salary ∈ [100, 10000]; *r* = 1000 employees **evenly distributed** among salaries:

sl(Salary ≥ 1000) = 0.90 (**90%**) *or s* = 900 employees

Combining Selectivity Estimations

- Given a *conjunctive query* Q *involving*: **A = *v* AND B** = ***u***: ***sl*(Q) = *sl*(A) · *sl*(B)** ∈ [0, 1]
- Given a *disjunctive query* Q *involving*: **A = *v* OR B** = ***u***: ***sl*(Q) = *sl*(A) +*sl*(B) - *sl*(A) · *sl*(B)** ∈ [0, 1]

```
SELECT * FROM EMPLOYEE WHERE  DNO = 5 AND Salary = 40000;
```

Assume: NDV(Salary) = 100, NDV(DNO) = 10, *r* = 1000 employees **evenly distributed** among salaries and departments:

sl(Salary) = 1/NDV(Salary) = 1/100 = 0.01

sl(DNO) = 1/NDV(DNO) = 1/10 = 0.1

sl(Q) = *sl*(Salary) · *sl*(DNO) = (1/10)·(1/100) = 0.001 or only *s* = 1 tuple

```
SELECT * FROM EMPLOYEE WHERE  DNO = 5 OR Salary = 40000;
```

Assume: NDV(Salary) = 100, NDV(DNO) = 10, *r* = 1000 employees **evenly distributed** among salaries and departments:

sl(Salary) = 1/NDV(Salary) = 1/100 = 0.01

sl(DNO) = 1/NDV(DNO) = 1/10 = 0.1

sl(Q) = (10/100) + (1/100) - (1/10)·(1/100) = 0.109 or *s* = 109 tuples

So Far…

Challenge: *Predict the number of tuples or blocks satisfying a selection*!

Assumption: The tuples are *uniformly distributed* across the values of attribute A

Selection Selectivity is: 1/NDV(A)
Selection Cardinality is: $r \cdot 1/\text{NDV}(A)$

For a **key** attribute, NDV(A) = *r* thus selection cardinality 1

Cost Estimation: Refinement

Query: `SELECT * FROM R WHERE R.A = v`

Context: *b* blocks, *f* blocking factor (tuples/block), *r* records

- **[S1] Linear Search: Expected Cost**: ***b*/2** and ***b*** if A is key and non-key, respectively.
- **[S2] Binary Search (R is sorted w.r.t. A):**
  - $\log_2(b)$ block accesses to reach the *first* block with record(s) A = *v*
  - If A is a **key**, then **Expected Cost: $\log_2(b)$ block accesses**
  - If A is **not a key**, then after accessing the first block, we have also to access *all linked* blocks whose records satisfy: A = *v*
  - *Selection cardinality* $s = r \cdot sl(A)$ tuples out of *r*
  - Blocking factor is *f* tuples/block: access **ceil(*s*/*f*)-1** more linked blocks:

Expected Cost: $\log_2(b)$ + ceil(s/f) -1 = $\log_2(b)$ + ceil($r \cdot sl(A)/f$) -1

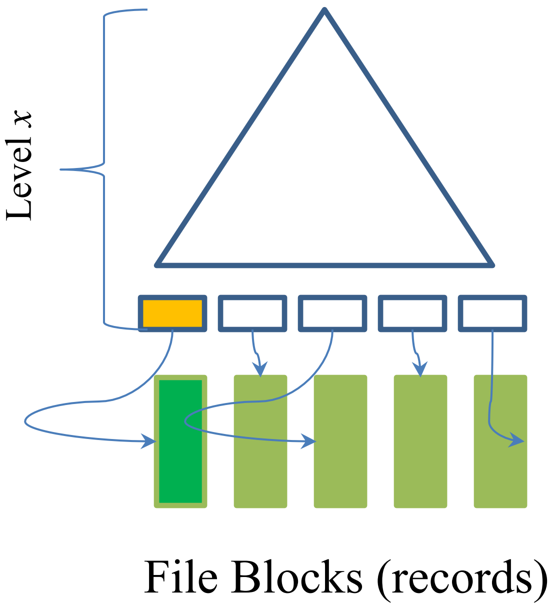
Note: If s =1, i.e., A is unique, $\log_2(b)$ + ceil(s/f) -1 = $\log_2(b)$ + 1 -1 = $\log_2(b)$.

Cost Estimation: Refinement

[S3] Primary Multilevel Index

- ISAM index of level: *x* over the ***primary key*** A *equality* A = *v*
- *One* block access *per* index level to reach the pointer to the data block plus 1 block for accessing the data block:

Expected Cost: *x* + 1



[S4] Hash File Structure

- Apply the hash function *h*(A) over the ***key*** A and retrieve the block.

Expected cost: 1

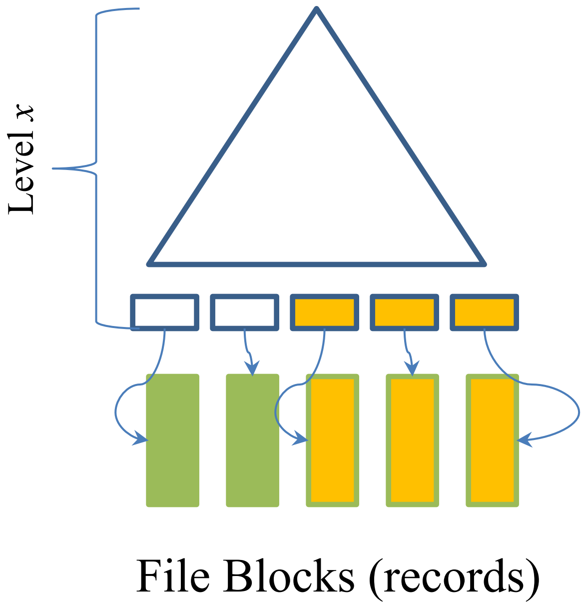
best case; *no* overflown buckets

Cost Estimation: Refinement

[S5] Primary Multilevel Index over ***primary key*** A in *range* search: <, >, <=, >=

- *One* block access *per* index level: ***x* block accesses.**
- Range selection cardinality: **s = *r* ·*sl*(*A*)** with ***sl*(*A*)** = *range* selectivity
- Blocking factor: *f* records/block
- **ceil(*s*/*f*) blocks for the *s* records**

Expected Cost: ***x* + ceil(*s*/*f*) = *x* + ceil(*r* ·*sl*(*A*) /*f*) block accesses.**

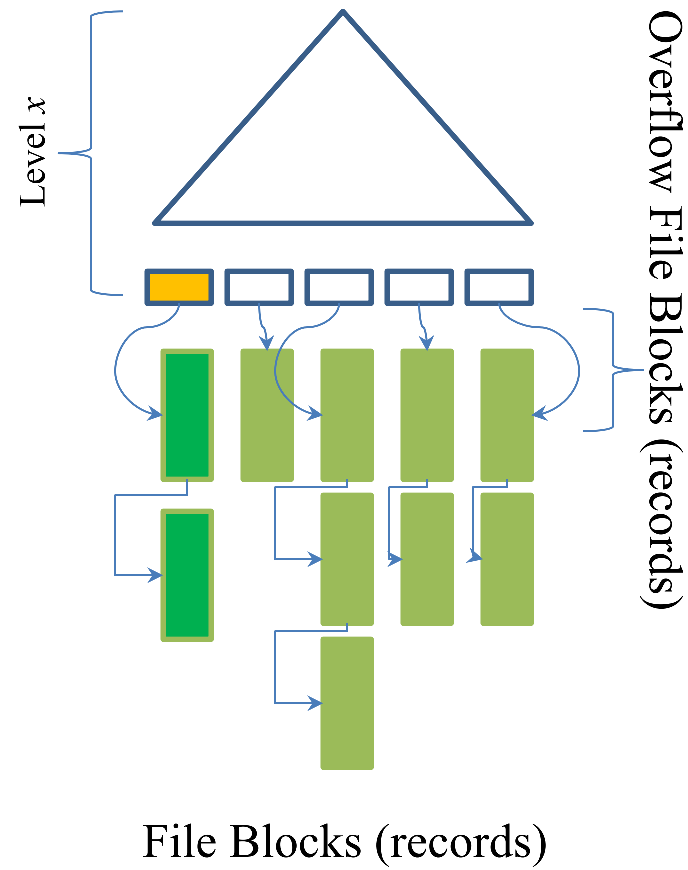


Cost Estimation: Refinement

[S6] Clustering Index over a ***non-key / ordering*** A in *equality* A = *v*

- *One* block access *per* index level: ***x* block accesses.**
- Selection cardinality ***s* = *r* ·*sl*(*A*) tuples**
- Blocking factor: *f* records/block
- **ceil(*s*/*f*) blocks for the *s* records**

Expected Cost: ***x* + ceil(*s*/*f*) = *x* + ceil(*r* ·*sl*(*A*) /*f*)**



Cost Estimation: Refinement

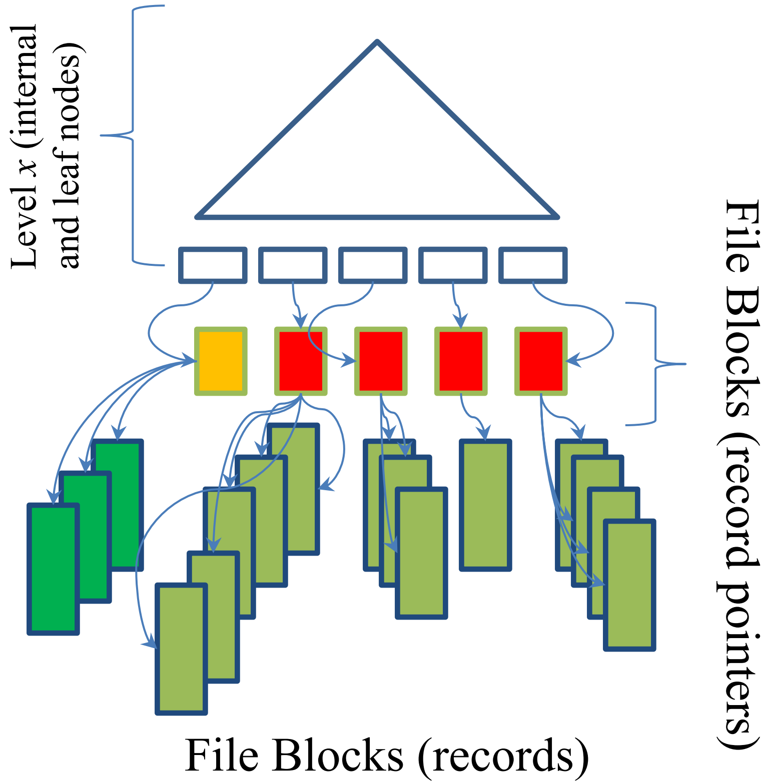
[S7] B+ Tree Secondary Index of *x* levels on attribute A for *equality* A = *v*

Note: the file is *not sorted* w.r.t. attribute A

Case 1: **A**ttribute A is **non-key**

- *One* block access *per* index level: ***x*** **block accesses.**
- Selection cardinality ***s* = *r* · *sl*(*A*) tuples**
- **1 block access** to *load* the block with data block pointers.
- Each tuple may be in a *different* data block (worst case) thus, access up to ***s*** blocks

Expected Cost: ***x* + 1 + *s* = *x* + 1 + *r* · *sl*(*A*)**



Cost Estimation: Refinement

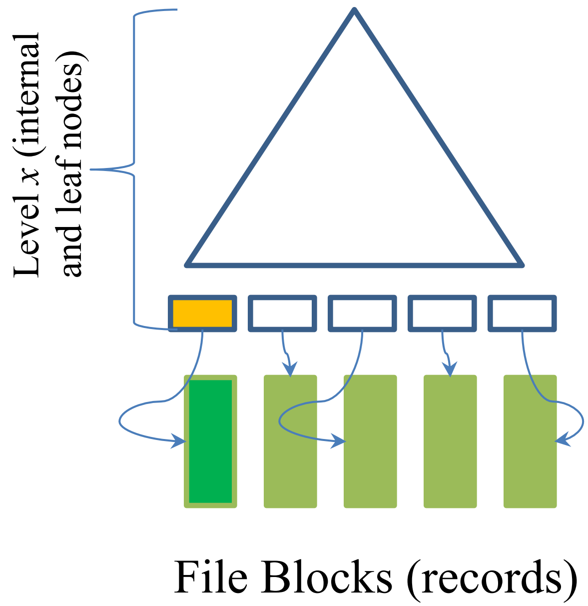
[S7] B+ Tree Secondary Index of *x* levels on attribute A for *equality* A = *v*

Note: the file is *not sorted* w.r.t. attribute A

Case 2: Attribute A is **key (unique)**

- *One* block access *per* index level: ***x* block accesses.**
- *One* data block since *s* = 1

Expected cost: ***x* + 1**



Example

Employee r = 10,000 records, b = 2,000 blocks, blocking factor f = 5 records/block

Access Paths:

- **Clustering Index on non-key Salary**: x_{Salary} = 3 levels, selectivity sl(Salary) = 0.002, indicating *average* cardinality s_{Salary} = 20 tuples.

- **Secondary Index on the key SSN:** x_{Ssn} = 4 levels, cardinality s_{Ssn} = 1 tuple, i.e., selectivity sl(Ssn) = 0.0001.

- **Secondary Index on non-key DNO:** x_{Dno} = 2 levels, L = 4 leaf blocks.
- NDV(DNO) = 125 distinct values, cardinality s_{Dno} = r/NDV(DNO) = 80 tuples, i.e., selectivity sl(DNO) = 1/125=0.008.

- **Secondary Index on non-key Gender:** x_{Gender} = 1 level.
- NDV(Gender) = 2 distinct values, cardinality s_{Gender} = 5000 tuples, selectivity sl(Gender) = 0.5

Note: Relation is sorted w.r.t. Salary due to the clustering index on Salary ☺

Example

[OP1] `SELECT * FROM EMPLOYEE WHERE Ssn = '123456789'`

- Linear Search: $b/2$ = **1,000 block accesses** (Ssn is *key*)
- Secondary Index (SSN): $x_{Ssn} + 1$ = **5 block accesses**

[OP2] `SELECT * FROM EMPLOYEE WHERE DNO > 5`

- Linear Search: b = **2,000 block accesses** (DNO is not a *key*)
- Secondary Index (DNO): $x_{Dno} + L/2 + r/2$ = **5,004 block accesses**
  - Worse than the linear search!

[OP3] `SELECT * FROM EMPLOYEE WHERE DNO = 5`

- Linear Search: b = **2,000 block accesses** (DNO is not a *key*)
- Secondary Index (DNO): $x_{Dno} + s_{Dno} + 1$ = **83 block accesses**

Example

[OP4] `SELECT * FROM EMPLOYEE WHERE DNO=5 AND SALARY>30000 AND GENDER = 'F'`

- Linear Search: *b* = **2,000 block accesses**

Evaluate the cost for *each* condition first:

- **Condition:** 'Dno = 5', best plan: **83 block accesses**
- **Condition:** 'SALARY > 30000', plan over clustering index: $x_{Salary} + b/2 = 3 + 1000 =$ **1,003 block accesses**
- **Condition:** 'Gender = 'F'', best plan: $x_{Gender} + s_{Gender} + 1 =$ **5,002 block accesses**

Strategy

- **Step 1:** apply selection 'Dno = 5' and retrieve **80 tuples** since cardinality of DNO = 80 tuples/employees per department!
  - Then filter out the most irrelevant records from the beginning!
  - 80 tuples or ceil(80/5) = 16 blocks may fit in memory!
- **Step 2:** apply the other *two* selections **in-memory** by just checking if: 'SALARY > 30000 AND GENDER = 'F''

Example

[OP5] `SELECT * FROM EMPLOYEE WHERE SALARY = 10000`

- Linear Search: b = **2,000 block accesses**
- Clustering Index (Salary): $x_{\text{Salary}} + \text{ceil}(s_{\text{Salary}}/f)$ = 3 + ceil(20/5) = **7 block accesses**

Join Selectivity & Cardinality

Consider: Join query: **R** $\bowtie_c$ **S** and Cartesian product: **R** × **S**

- `SELECT * FROM R, S`
- `SELECT * FROM R, S WHERE R.A = S.B`

with joining condition: $C := \{R.A = S.B\}$

Definition 1: *join selectivity* (*js*) is the fraction of the matching tuples (between the relations **R** and **S**) out of the Cartesian product:

$$js = |\mathbf{R} \bowtie_c \mathbf{S}| / |\mathbf{R} \times \mathbf{S}| \text{ with } 0 \leq js \leq 1.$$

Cardinality of Cartesian product: $|\mathbf{R} \times \mathbf{S}| = |\mathbf{R}| \cdot |\mathbf{S}|$

Definition 2: *join cardinality jc* $:= js \cdot |\mathbf{R}| \cdot |\mathbf{S}|$

Challenge: Predict the join cardinality (*jc*), i.e., the *size* of the join result, without executing the join query.

Join Selectivity & Cardinality

```
SELECT * FROM EMPLOYEE E, DEPARTMENT D
```

```
SELECT * FROM EMPLOYEE E, DEPARTMENT D
WHERE    E.SSN = D.MGR_SSN
```

- |**D**| = 10 departments, i.e., 10 managers,
- |**E**| = 1000 employees,

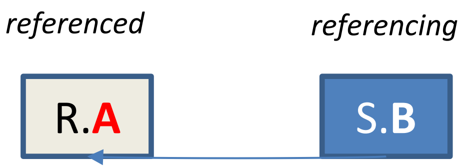
- Cardinality of Cartesian product: |**D**|·|**E**| = 10 ·1000 = **10,000 tuples**
- Join cardinality ***jc* = 10** matching tuples, **why?**

- **Because:** each department has *only one* manager;
- Join selectivity ***js* = 10/(10·1000) = 0.001** (0.1% matching tuples)
- *Probability of selecting a matching tuple in the Cartesian space*

Join Selectivity

Focus: *equijoin* **R**.A = **S**.B

- Attribute A is the **primary key**
- Attribute B is the **foreign key** from **S** to **R**. Then:



$$|\mathbf{R} \bowtie_{c} \mathbf{S}| \leq |\mathbf{S}|$$

Why?

Each tuple of **S** *will* match with *zero* or at most *one* tuple from **R**. Hence,

$$js \leq |\mathbf{S}|/|\mathbf{R} \times \mathbf{S}| = |\mathbf{S}|/|\mathbf{R}| \cdot |\mathbf{S}| = 1/|\mathbf{R}|$$

$$jc := js \cdot |\mathbf{R}| \cdot |\mathbf{S}| = |\mathbf{S}|$$

A department tuple (D) matches with *only* one employee (E), i.e., the manager.

```
SELECT * FROM EMPLOYEE E, DEPARTMENT D
WHERE    E.SSN = D.MGR_SSN
```

- |D| = 10 departments; |E|=1000 employees
- ***js* = 1/1000; *jc* = 10**

Join Selectivity

Theorem 1. Given n = NDV(A, **R**) and m = NDV(B, **S**):

$$js = 1 / \max(n, m)$$

$$jc = (|\mathbf{R}| \cdot |\mathbf{S}|) / \max(n, m)$$

Proof. *Beyond the scope of the lecture…*

Example: **Show the dependents of each employee; an employee might have zero to many dependents**

```
SELECT * FROM EMPLOYEE E, DEPENDENT P
WHERE    E.SSN = P.E_SSN
```

n = NDV(SSN,**E**) = 2000 employees

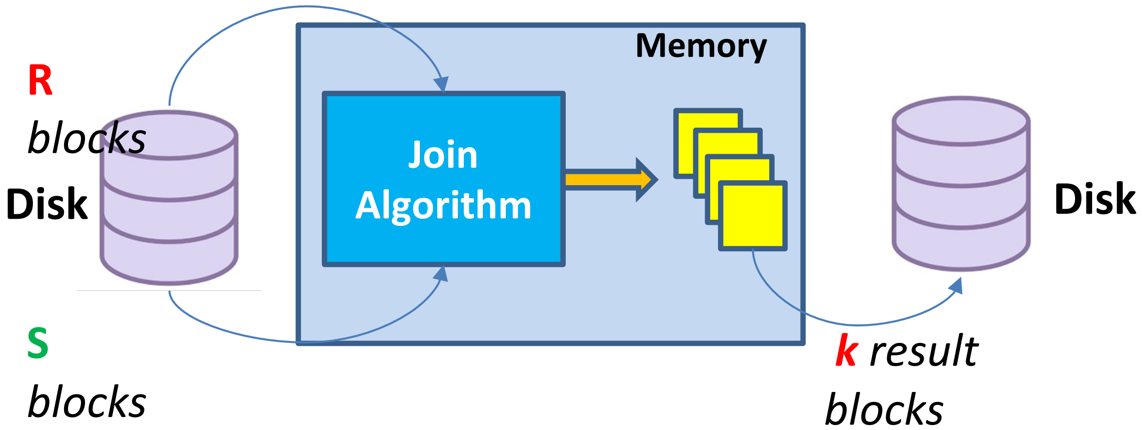
m = NDV(E_SSN, **P**) = 40 dependents

js = 1/max(2000,40) = 1/2000 = 0.0005 or **0.05%**

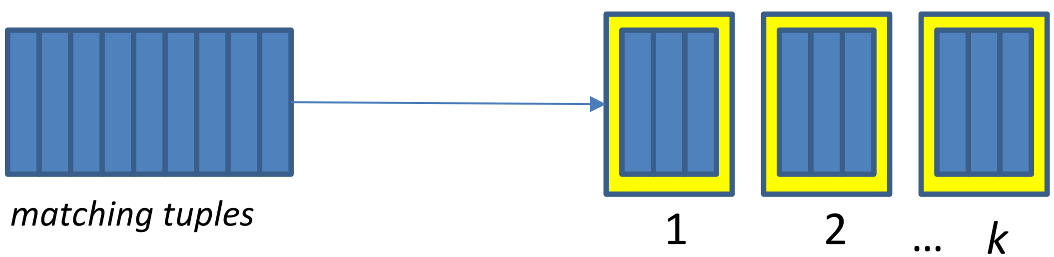
jc = 0.0005 * 2000 * 40 = 40 matching tuples

Join Selectivity

- Relation **R** and **S** with b_R and b_S blocks; $\mathbf{R}.A = \mathbf{S}.B$
- Memory: n_B blocks; $n = \text{NDV}(\mathbf{R}.A)$, $m = \text{NDV}(\mathbf{S}.B)$
- Result block: *blocking factor* f_{RS} **tuples/block**



- *Write* every *full result block* to disk. **How many result blocks do we *write*?**
- Matching tuples: $jc = js \cdot |\mathbf{R}| \cdot |\mathbf{S}| = (1/\max(n, m)) \cdot |\mathbf{R}| \cdot |\mathbf{S}|$
- #result blocks: $k = (js \cdot |\mathbf{R}| \cdot |\mathbf{S}|) / f_{RS}$



Join Cost Estimation: Refinement

Nested-Loop Join

```
SELECT * FROM EMPLOYEE E, DEPARTMENT D
WHERE    E.SSN = D.MGR_SSN
```

- **D** is the *outer* relation, i.e., |**D**|< |**E**| with outer loops: **ceil(b_D/(n_B-2))**

Estimated Cost: $b_D + (\mathrm{ceil}(b_D/(n_B-2)) \cdot b_E$

- Matching tuples: $jc = js \cdot |\mathbf{E}| \cdot |\mathbf{D}| = (1/\max(n, m)) \cdot |\mathbf{E}| \cdot |\mathbf{D}|$
- Number of result blocks: $k = (js \cdot |\mathbf{E}| \cdot |\mathbf{D}|) / f_{RS}$

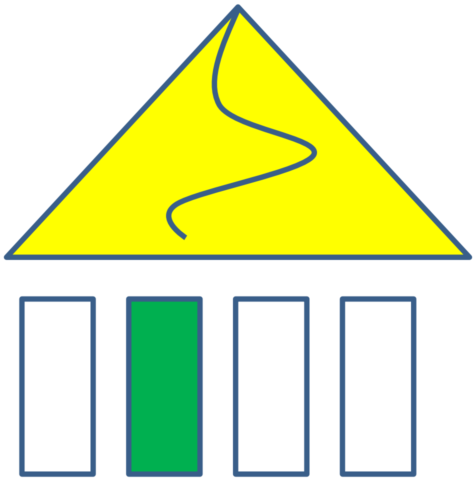
Refined Estimated Cost: $b_D + (\mathrm{ceil}(b_D/(n_B-2)) \cdot b_E + (js \cdot |\mathbf{E}| \cdot |\mathbf{D}| / f_{RS})$

Join Cost Estimation: Refinement

Index-based Nested-Loop Join

Index on attribute **`MGR_SSN`** x_D levels

```
SELECT * FROM EMPLOYEE E, DEPARTMENT D
WHERE    E.SSN = D.MGR_SSN
```



For *each* employee *e*, use index to *find* matching manager: *e*.**SSN** = *d*.**MGR_SSN**

Matching tuples: $jc = js \cdot |E| \cdot |D| = (1/\max(n, m)) \cdot |E| \cdot |D|$

Number of result blocks: $k = (js \cdot |E| \cdot |D|) / f_{RS}$

Case: Primary Multilevel Index on **`MGR_SSN`**: **key / ordering**:

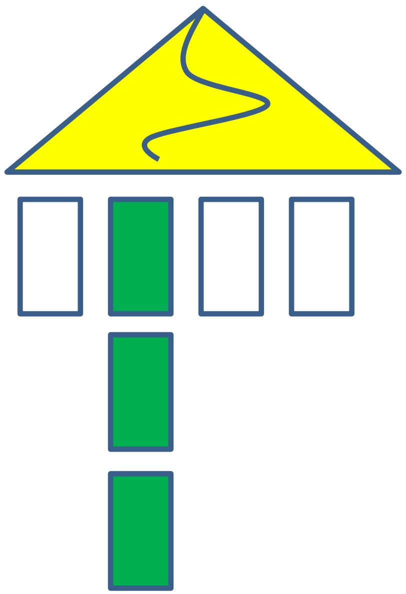
Refined Cost: $b_E + |E| \cdot (x_D + 1) + (js \cdot |E| \cdot |D| / f_{RS})$

Join Cost Estimation: Refinement

Index-based Nested-Loop Join

Index on attribute DNO x_E levels, selection cardinality s_E, blocking factor f_E

```
SELECT * FROM EMPLOYEE E, DEPARTMENT D
WHERE    E.DNO = D.DNUMBER
```



- Selection cardinality of DNO $s_E = (1/\mathrm{NDV}(\mathrm{DNO}))\cdot|\mathbf{E}|$

For *each* department *d*, use index to *find* employee

e.DNO = *d*.**DNUMBER**

Case: Clustering Index on DNO: **non-key / ordering**

- For *each* department, *load* the corresponding employees.
- Selection cardinality := employees per department: s_E
- Blocks of employees per department: s_E / f_E
- Number of result blocks: $k = (js \cdot |\mathbf{E}| \cdot |\mathbf{D}|) / f_{RS}$

- **Refined Cost:** $b_D + |\mathbf{D}| \cdot (x_E + s_E/f_E) + (js \cdot |\mathbf{E}| \cdot |\mathbf{D}| / f_{RS})$

Join Cost Estimation: Refinement

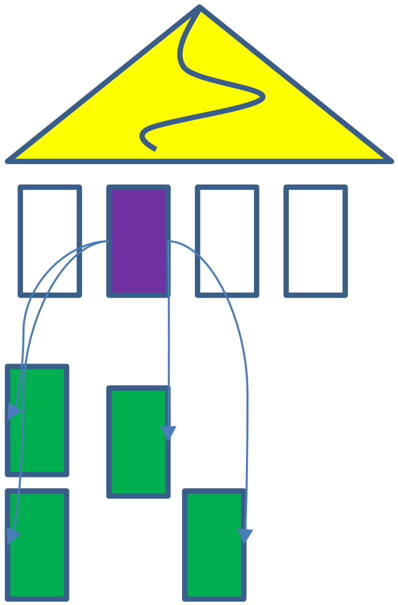
Index-based Nested-Loop Join

Index on attribute **DNO** x_E levels, selection cardinality s_E, blocking factor f_E

```
SELECT * FROM EMPLOYEE E, DEPARTMENT D
WHERE    E.DNO = D.DNUMBER
```

- Selection cardinality of DNO $s_E = (1/\text{NDV(DNO)}) \cdot |E|$

For *each* department *d*, use the index to *find* employee *e*.**DNO** = *d*.**DNUMBER**



Case: B+ Tree Secondary Index on **DNO**: **non-key / non-ordering**

- For *each* department, *load* the corresponding employees.
- **1 block** (block of pointers) + **blocks** of employees: s_E
  - each employee belongs to a different block (worst case)
- Number of result blocks: $k = (js \cdot |E| \cdot |D|) / f_{RS}$

Refined Cost: $b_D + |D| \cdot (x_E + 1 + s_E) + (js \cdot |E| \cdot |D| / f_{RS})$

Join Cost Estimation: Refinement

Sort-Merge: *both files are sorted* on attributes A and B

- **Refined Cost:** $b_R + b_S + (js \cdot |R| \cdot |S| / f_{RS})$

Hash-Join: *both files are hashed* w.r.t. same hash function *h*;

- **Refined Cost:** $3 \cdot (b_R + b_S) + (js \cdot |R| \cdot |S| / f_{RS})$

Example

Employee r_E = 10,000 records, b_E = 2,000 blocks.

Department r_D = 125 records, b_D = 13 blocks.

Result block: f_{RS} = 4 records/block (blocking factor).

Memory: n_B = 10 blocks

```
SELECT * FROM EMPLOYEE E, DEPARTMENT D WHERE E.DNO = D.DNUMBER
```

Access Paths:

- Primary Index on **DNUMBER** $x_{Dnumber}$ = 1 level.
- Secondary Index on **DNO** x_{Dno} = 2 levels.
- Selectivity *sl*(DNO) = 1/ NDV(DNO) = 1/125 = 0.008.
- Selection cardinality s_{Dno} = *sl*(DNO) * r_E = 80 employees per department.

Task: Expected cost of the JOIN query involving *selection* & *join* selectivities.

Example

Join selectivity & join cardinality:

- js = 1/ max(NDV(**DNO**), NDV(**DNUMBER**))
- n = NDV(**DNO**) = 125; m = NDV(**DNUMBER**) = 125
- js = 1/max(125,125) = 1/125 = 0.008

- $jc = js * r_E * r_D$ = **10,000 matching tuples**
- Number of result blocks: $k = jc/f_{RS}$ = 10,000/4 = **2,500 result blocks**

Example

Nested-loop Join:

- Department *outer*: $b_D + (\text{ceil}(b_D/(n_B\text{-}2)) \cdot b_E + (js \cdot r_E \cdot r_D / f_{RS})$ = **6,513 block accesses**

Index-based Nested-loop Join *with* Employee as outer

Primary Index (DNUMBER): $b_E + (r_E \cdot (x_{Dnumber}+1)) + (js \cdot r_E \cdot r_D / f_{RS})$ = **24,500 block accesses**

Index-based Nested-loop Join *with* Department as outer

Secondary Index (DNO): $b_D + (r_D \cdot (x_{Dno} + s_{Dno} + 1)) + (js \cdot r_E \cdot r_D / f_{RS})$ = **12,888 block accesses**

Hash-Join: $3 \cdot (b_D + b_E) + (js \cdot r_E \cdot r_D / f_{RS})$ = **8,539 block accesses**

Sort-Merge: **Cannot be used.** *Why?*

Best strategy: **Nested-loop Join** with Department *outer*

Observation: indexing methodology is not a *panacea* for query processing!

Optimization Plan Example

Employee: r = 10,000 tuples, b = 2,000 blocks, f = 5 records/block.

Memory: 100 available blocks

Access Paths:

- **Clustering Index** on Salary (non-key/ordering) x_{Salary} = 3 levels.
- NDV(Salary) = 500.

- **B+ Tree Secondary Index** on DNO (non-key/non-ordering) x_{Dno} = 2 levels.
- NDV(DNO) = 125.

```
SELECT * FROM EMPLOYEE
WHERE    DNO = 5 AND Salary = 1000
```

Optimization Plan Example

Plan 1:

- First, evaluate the selection: `SELECT * FROM EMPLOYEE WHERE DNO = 5`
- Then, given this *intermediate* result, evaluate the selection: `Salary = 1000`.

Plan 2:

- First, evaluate: `SELECT * FROM Employee WHERE Salary = 1000`
- Then, given this *intermediate* result, evaluate the selection: `DNO = 5`.

- **Task:** Calculate the cost for Plan 1 and Plan 2 and *decide* on the best plan.

Optimization Plan 1

Plan 1:

Evaluate: `SELECT * FROM EMPLOYEE WHERE DNO = 5`

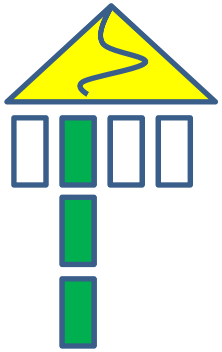
Exploit: B+ Tree on DNO with x_{Dno} = 2 levels, NDV(DNO) = 125

- B+ Tree navigation: **2 block accesses.**
- DNO: non-key, thus, retrieve the block of pointers to records satisfying DNO = 5.
- Selection cardinality $s_{Dno} = r$/NDV(DNO) = 80 tuples (*sl*(DNO) = 1/125 = 0.008).
- **80 block accesses**; each block is accessed corresponding to record with DNO=5.

Total Cost: 2 blocks + 1 block of pointers + 80 blocks = **83 block accesses.**

- Intermediate result size: cardinality is 80 tuples or ceil(80/5) = **16 blocks**
- Result fits in memory (up to 100 blocks).
- Then, search over 16 blocks *in-memory* for: `Salary = 1000`.

Optimization Plan 2



Plan 2:

Evaluate: `SELECT * FROM Employee WHERE Salary = 1000`

Exploit: Clustering Index on Salary, x_{Salary} = 3 levels, NDV(Salary) = 500.

- Clustering Index to point to the first block of cluster Salary = 1000: **3 block accesses.**
- Selectivity *sl*(Salary) = 0.002 and cardinality $s_{\text{Salary}} = r \cdot sl(\text{Salary})$ = 20 tuples.
- Each cluster occupies ceil(s_{Salary} /*f*) = ceil(20/5) = 4 blocks of records with Salary = 1000.

Total Cost: 3 blocks + 4 cluster blocks = **7 block accesses.**

- Intermediate result size: **4 blocks**
- Result fits in memory (up to 100 blocks).
- Then, search over 4 blocks *in-memory* for: `DNO = 5`.

Plan 2 is the best!

University of Glasgow | School of Computing Science

Special Thanks

Special Thanks to Dr Nikos Ntarmos who is the original author of the slides.