# Some Exam Questions 2018-2021

Database Systems DB(H)

Dr Chris Anagnostopoulos

# EXAM QUESTION 3 (2021) 1/6

EMPLOYEE(<u>SSN</u>, Salary, Age, DNO),

- **Clustering Index** Age *non-key ordering* with $t$ levels.
- **B+ Tree** DNO *non-key non-ordering* with $x_{DNO} = 3$ levels.
- **B+ Tree** Salary *non-key non-ordering* with $x_{Salary} = 3$ levels.
- NDV(DNO) = 100; NDV(Salary) = 500; NDV(Age) = 200.
- $r$ = 10000 tuples of size R = 250 bytes (DNO = 50; Salary = 100, Age = 50, SSN = 50), B = 512 bytes, any pointer V = 50 bytes. Available memory: 200 blocks.
- **Assumption**: all the values of each attribute are uniformly distributed across

```
SELECT * FROM  EMPLOYEE
WHERE (AGE = 60 OR AGE = 25) AND (DNO = 5)
      AND (Salary = 60000 OR Salary = 30000)
```

**(a).** Level $t$ of the multi-level Clustering index over Age.

**(b).** Which is the *selection selectivity* of the query?

**(c).**<u>**Three**</u> *query processing plans* and select the *best*

# EXAM QUESTION 3 (2021) 2/6

**(a).** Level *t* of the multi-level Clustering index over Age.

- **NDV(Age) = 200**. *r* = 10000 tuples of size R = 250 bytes (DNO = 50; Salary = 100, Age = 50, SSN = 50), **B = 512 bytes**, any pointer V = 50 bytes.
- Blocking factor of the data file is floor(B/R) = 2, thus, there are *b* = ceil(*r*/bfr) = 5000 blocks in the file.
- File is *sorted* w.r.t. Age. Index entry size of Age + Pointer = **100 bytes**.
- Blocking factor of index: ibfr = **5 entries/block**.
- **Fact**: we need as many index entries as the number of Age's distinct values, i.e., 200 index entries, which should be stored in 200/5 = **40 blocks**.

- **Level 1**: Clustering index has **40 blocks.**
- **Level 2**: It is a Primary index, which needs as many entries as the number of blocks of Level 1, i.e., 40 index entries stored in 40/5 = **8 blocks.**
- We need to *keep* building levels until reaching the level that has only *one* block.
- **Level 3** has 8 entries are stored in **2 blocks**.
- **Level 4** has 2 entries stored in **one** only block!

- 4 levels. L1: 40 blocks, L2: 8 blocks, L3: 2 blocks, L4: 1 block.

# EXAM QUESTION 3 (2021) 3/6

**(b).** Which is the *selection selectivity* of the query?

```
SELECT * FROM  EMPLOYEE
WHERE (AGE = 60 OR AGE = 25) AND (DNO = 5)
      AND (Salary = 60000 OR Salary = 30000)
```

- NDV(DNO) = 100; NDV(Salary) = 500; NDV(Age) = 200.
- Sel(Age) = r/NDV(Age) = 50; Sel(Salary) = r/NDV(Salary) = 20

- **sel(AGE = 60 OR AGE = 25)** = sel(60)+sel(25) = 50 + 50 = **100 tuples** (**0.01**)
- **sel(DNO=5) = 100 (0.01)**
- **sel(Salary = 60000 OR Salary = 30000)** = sel(60K) + sel(30K) = 20 + 20 = **40** (**0.004**)
- *mutual exclusivity over the Age and Salary values.*
- Selection cardinality: (0.01*0.01*0.004)*10000 = **0.04 tuples!**
- Selection selectivity: 0.01*0.01*0.004.

# Exam Question 3 (2021) 4/6

**(c). <u>Three</u>** *query processing plans* and select the *best*

```
SELECT * FROM  EMPLOYEE
WHERE (AGE = 60 OR AGE = 25) AND (DNO = 5)
      AND (Salary = 60000 OR Salary = 30000)
```

- Sub-query: **Age= 30 OR 50**: We go down the index in two *different* paths.
- For a specific age: **4 block accesses** to go down the index before accessing the cluster.
  - Age is uniformly distributed, i.e., we have 10000/200 = 50 employees/age
  - For a cluster of employees with same age: 50/2 = 25 blocks.
  - **Cost** per Age: 4 + 25 = **29 block accesses**.
  - In total: **29 + 29 = 58 block accesses**.
- Sub-query: **DNO = 5**: Use of B+ Tree and number of blocks with block pointers pointing to blocks with at least one employee working in Department 5.
  - Going down the B+ tree to find the leaf node corresponding to DNO = 5
  - Pointer to a *cluster* of blocks with block-pointers: we can store floor(B/V) = 10 pointers/block.
  - DNO is uniformly distributed, i.e., there are 10000/100 = 100 employees/department.
  - We need to store 100 block-pointers stored into 100/10 = 10 blocks.
  - **Cost**: B+ Tree (3 blocks) + 10 block accesses for accessing blocks with pointers + for *each* block we access the 10 actual data blocks.
  - In total: **3 + 10 + 10*10 = 113 block accesses.**

# Exam Question 3 (2021) 5/6

**(c).** **<u>Three</u>** *query processing plans* and select the *best*

```
SELECT * FROM  EMPLOYEE
WHERE (AGE = 60 OR AGE = 25) AND (DNO = 5)
      AND (Salary = 60000 OR Salary = 30000)
```

- Sub-query: **Salary 60K or 30K**. We access the B+ tree *twice* due to *disjunction*.
  - For Salary = 30K, we need to know the number of employees having this salary.
  - Under uniformity: we obtain 10000/500 = 20 employees/salary.
  - That is, we need to access these 20 data blocks, *each* one corresponding to an employee with Salary = 30K, thus, we need 20 block pointers stored into **2 blocks of block-pointers**.
  - **Cost**: B+ Tree (3 blocks) + access 2 blocks of block-pointers + for *each* of them, load the corresponding 10 actual blocks.
  - Cost per Salary value: 3 + 2 + 2*10 = 25 block accesses *per* Salary.
  - In total: for two salary queries: **25 + 25 = 50 block accesses**.

# Exam Question 3 (2021) 6/6

**(c). Three** *query processing plans* and select the *best*

```
SELECT * FROM  EMPLOYEE
WHERE (AGE = 60 OR AGE = 25) AND (DNO = 5)
      AND (Salary = 60000 OR Salary = 30000)
```

- **Policy 1:** First **(AGE = 60 OR AGE = 25)** with 58 block accesses. Then we have retrieved 50 + 50 = **100 employees** with ages 60 and 25.
  - We need memory of 50 blocks to store them (2 employees per block). Our memory is sufficient, thus with a linear scan we go through the other conditions. **[58 accesses; memory 50 blocks]**
- **Policy 2:** First **(DNO = 5)** with 113 block accesses. Then we have retrieved 100 employees working in DNO=5.
  - We need memory of 100/2 = 50 blocks to store them. Our memory is sufficient, thus with a linear scan we go through the other conditions. **[113 accesses; memory 50 blocks]**
- **Policy 3:** First (**Salary = 60000 OR Salary = 30000)** with 50 block accesses. We retrieve 20 employees per salary or 40 employees for the two salary values, needing memory of 40/2 = 20 blocks **[50 accesses; memory 20 blocks].**
- **Policy 3 is the BEST w.r.t. number of block accesses!**

# EXAM QUESTION 4 (2021) 1/5

**EMPLOYEE(<u>SSN</u>, NAME, SALARY);**

**DEPENDENT(<u>ESSN, DEPENDENT_NAME</u>)**

$r$ = 500 employees and $d$ = 30 tuples in DEPENDENT.

Size of each attribute: 10 bytes; B = 64 bytes; any pointer V = 10 bytes

$m$ = NDV(ESSN) = 10. ESSN is uniformly distributed.

```
SELECT E.NAME, D.DEPENDENT_NAME
FROM   EMPLOYEE AS E, DEPENDENT AS D
WHERE  E.SSN = D.ESSN
```

- **Clustering Index** over ESSN in DEPENDENT.
- **B+ Tree** over SSN with $x_{SSN}$ = 3 levels in EPMPLOYEE.

**Task 1:** *join cardinality* and *blocking factor* of the result block (number of matching tuples *per* block.)

**Task 2:** Provide <u>**two**</u> *join processing methods* using the <u>available</u> access paths and select the *best* one in terms of block accesses.

# EXAM QUESTION 4 (2021) 2/5

**EMPLOYEE(<u>SSN</u>, NAME, SALARY);**

**DEPENDENT(<u>ESSN, DEPENDENT_NAME</u>)**

$r$ = 500 employees and $d$ = 30 tuples in DEPENDENT.

Size of each attribute: 10 bytes; B = 64 bytes; any pointer V = 10 bytes

$m$ = NDV(ESSN) = 10. ESSN is uniformly distributed.

**`SELECT E.NAME, D.DEPENDENT_NAME`** …

- *blocking factor* of the result block (the number of matching tuples *per* block.):

SELECT clause: the size of the matching tuple is: 10 bytes (E.Name) + 10 bytes (Dependent_Name) = 20 bytes.

rbfr = floor(64/20) **= 3 matching tuples per block** .

- *join cardinality* js = 1/(max(NDV(SSN),NDV(ESSN))) * $r*d$ = (1/500)*500*30 = 30 matching tuples, which is expected, since there are **30 tuples in DEPENDENT** !

# EXAM QUESTION 4 (2021) 3/5

$r$ = 500 employees and $d$ = 30 tuples in DEPENDENT.

Size of each attribute: 10 bytes; B = 64 bytes; any pointer V = 10 bytes

$m$ = NDV(ESSN) = 10. ESSN is uniformly distributed.

**`SELECT`** … **`WHERE  E.SSN = D.ESSN`**

**Task 2:**

**Employee**: record size = R = 30 bytes; bfr = floor(64/30) = 2 employees/block, thus, there are ceil(500/2) = **bE = 200 blocks** .

**Dependent**: record size = R' = 20 bytes; bfr = floor(64/20) = 3 dependents/block, thus, there are ceil(30/3) = **bD = 10 blocks**.

Number of result blocks required to be *written* from the memory to the disk during the join process is: number of matching tuples out of the blocking factor of the result block, i.e., $k$ = js/rbfr = 30/3 = 10 blocks, that is, **10 block accesses (write).**

Clustering index on ESSN has:

- Index entry size: 10 bytes + 10 bytes = 20 bytes; floor(64/20) = 3 entries per block. We need 10 entries, one per ESSN, thus the clustering index is **4 blocks**. Search within the clustering index is **log2(4) = 2 block accesses** .

# EXAM QUESTION 4 (2021) 4/5

*r* = 500 employees and *d* = 30 tuples in DEPENDENT.

Size of each attribute: 10 bytes; B = 64 bytes; any pointer V = 10 bytes

*m* = NDV(ESSN) = 10. ESSN is uniformly distributed.

**`SELECT … WHERE  E.SSN = D.ESSN`**

- **Method J1:** Index-based Loop Join using the Clustering Index over ESSN. That is, for each employee, we check if we can find a matching cluster of their dependents. There are 10 clusters of dependents, since they refer to 10 different employees. Then, since there are 30 dependents, based on the uniformity assumption, **each cluster has 3 dependents,** thus, **each cluster has 1 block.**

- **Cost J1**: bE + *r* · (2 block accesses + 1 block access) + 10 block accesses = 200 + 500 (2 + 1) = **1700 block accesses**.

(*all blocks from employee; we search r times, one per employee, in order to search if an employee has dependents. We access the clustering index and retrieve all the dependents of the employee + the total cost of writing the result blocks.)*

# EXAM QUESTION 4 (2021) 5/5

$r$ = 500 employees and $d$ = 30 tuples in DEPENDENT.

Size of each attribute: 10 bytes; B = 64 bytes; any pointer V = 10 bytes

$m$ = NDV(ESSN) = 10. ESSN is uniformly distributed.

```
SELECT … WHERE  E.SSN = D.ESSN
```

**Method J2:** Index-based Loop Join using the B+ Tree on SSN.

For each dependent, we search for their corresponding employee.

Cost J2: bD + $d$ *(3 + 1) + 10 = 10 + 30*4 + 10 = **140 block accesses**.

**This is the best method: using the B+Tree on SSN.**

# EXAM QUESTION 3 (2020) 1/4

EMPLOYEE(SSN, Salary, DNO),

- $r$ = 10,000 employees; $b$ = 5000 blocks, $f$ = 2 records/block; memory 2,000 blocks.
- **Clustering Index: Salary** non-key ordering $x_{\text{Salary}}$ = 3 levels; range: from 10,000 to 100,000.
- **B+ Tree Index: DNO** non-key non-ordering $x_{\text{DNO}}$ = 2 levels. NDV(DNO) = 5.
- We need only 1 block of data block-pointers per DNO value.

```
SELECT * FROM EMPLOYEE
WHERE (10000 <= Salary) AND (Salary <= 40000)
      AND (DNO = 1 OR DNO = 3)
```

**Task 1:** Estimate the selection cardinality.

**Task 2:** Propose **two** *query processing plans* and select the *best*.

# Exam Question 3 (2020) 2/4

```
SELECT * FROM EMPLOYEE
WHERE (10000 <= Salary) AND (Salary <= 40000)
      AND (DNO = 1 OR DNO = 3)
```

**Task 1:** Conjunction of conditions: selection selectivity as a product of:
sel = sel(DNO =1 or DNO = 3}) * sel(Salary in [10000, 40000]).

Disjunction of mutual exclusion: sel(DNO) = 1/$n$ + 1/$n$ = 0.4 (DNO is either 1 or 3)
Range: sel(Salary) = (40000-10000)/(max(Salary)-min(Salary))) = 1/3 = 0.33.

Selectivity = 0.33*0.4 = 0.132.
Selection cardinality = selectivity * number of employees = 1320 employees.

# Exam Question 3 (2020) 3/4

**Task 2:** Two possible plans (two conditions).

**Plan 1:** Execute DNO is 1 or 3, then, check for: 40000 >= Salary >= 10000 (intermediate results).

**Plan 2:** The opposite.

**Plan 1:** Use B+ Tree for DNO = 1. Selection cardinality s = $r$*sel(DNO = 1) = $r/n$ = 2000 tuples per DNO.

- *one* block access per level (2 block accesses).
- *one* block access to load the block of block pointers.
- Employees may be in different data blocks (worst case) thus, access up to s = 2000 blocks.
- Cost(DNO=1): 2 + 1 + 2000 = 2003 block accesses;
- Cost(DNO=3): 2 + 1 + 2000 = 2003 (since, different blocks can be fetched)
- Selection over DNO: 4006 block accesses; we retrieve 2000+2000 = 4000 employees stored in ceil(4000/2) = 2000 blocks in-memory (available).

Filter out irrelevant tuples in memory.

**Plan 1 Cost: 4006 block accesses; memory: 2000 blocks.**

# Exam Question 3 (2020) 4/4

**Plan 2:** Execute: 40000 >= Salary >= 10000, then, check for DNO is 1 or 3.

Use Clustering Index on Salary:

- 3 block accesses in the index;
- Selection cardinality s = *r**sel(Salary in [10K, 40K] = 0.33*10000 = 3300 employees, fitting in ceil(3300/2) = 1650 blocks retrieved **contiguously**
- Cost: 1653 block accesses retrieving the 3300 tuples, stored in 1650 blocks in-memory (available)

Filter out irrelevant tuples in memory.

**Plan 2 Cost: 1653 block accesses ; memory: 1650 blocks.**

**Best plan: Plan 2**

# Exam Question 4 (2020) 1/4

EMPLOYEE(<u>SSN</u>, Salary)

- $r$ = 5600 employees; each attribute has size 7 bytes, B = 64 bytes; P = 7 bytes; $n$ = NDV(Salary) = 280, salary is *uniformly* distributed.
- **B+ Tree of level *t*** on non-ordering, non-key Salary: leaf node fits in 1 block.

**Task 1:** Calculate the leaf node order $q$: $q$-1 index values, $q$-1 block pointers, and *one* sibling leaf pointer

**Task 2:** How many leaf nodes does the B+ Tree have?

```
SQL1:SELECT * FROM EMPLOYEE WHERE Salary = 50000
```

**Task 3:** Express the expected cost using the B+ Tree as a function of level $t$.

```
SQL2:SELECT * FROM EMPLOYEE
     WHERE Salary = 50000 OR Salary = 60000
```

**Task 4:** Express the expected cost using the B+ Tree as a function of level $t$.

# Exam Question 4 (2020) 2/4

EMPLOYEE(<u>SSN</u>, Salary)

- $r$ = 5600 employees; each attribute size 7 bytes, B = 64 bytes; P = 7 bytes; $n$ = NDV(Salary) = 280, salary is *uniformly* distributed.
- **B+ Tree of level *t*** on non-ordering, non-key Salary: leaf node fits in 1 block.

**Task 1:** Calculate the leaf node order $q$: $q$-1 index values, $q$-1 block pointers, and *one* sibling leaf pointer

All the 280 distinct values of Salary should be stored in leaf nodes.

- 1 leaf node fits in 1 block;
- Order $q$: ($q$-1) index values + ($q$-1) data pointers + 1 sibling pointer.

  $(q\text{-}1)*7 + (q\text{-}1)*7 + 7 <= 64$ bytes, or $q$ -1 <= 4.07 or $q$ <= 5.07.

  That is, **Order *q* = 5**

**Task 2:** Order $q$ = 5, i.e., each leaf node stores $q$-1 = 4 distinct salary values, thus, we need 280/4 = **70 leaf nodes**.

# Exam Question 4 (2020) 3/4

EMPLOYEE(SSN, Salary)

- $r$ = 5600 employees; each attribute size 7 bytes, B = 64 bytes; P = 7 bytes; $n$ = NDV(Salary) = 280, salary is *uniformly* distributed.
- **B+ Tree of level *t*** on non-ordering, non-key Salary: leaf node fits in 1 block.

```
SQL1:SELECT * FROM EMPLOYEE WHERE Salary = 50000
```

**Task 3:** 4 Salary values (distinct) per leaf node.

- *Uniformity*: there are 5600/280 = 20 employees having the same Salary.
- For each pointer in leaf node (2-level indirection), we point to blocks that store up to 20 pointers to data blocks containing *at least* one employee with the indexed salary value.
- There are 20 employees per Salary, thus, 20 block pointers (worst case).
- One block accommodates floor(64/7) = 9 block pointers (we need 20 pointers).
- For each Salary, we need **3 blocks** with block pointers: 2 blocks with 9 pointers each, and one block with 2 pointers.
- **Cost**: $t$ block accesses to reach the leaf node + 3 block accesses to load the 3 blocks with block pointers and then 20 block accesses to actual data blocks.
- **In total: *t* + 23 block accesses**

# Exam Question 4 (2020) 4/4

EMPLOYEE(SSN, Salary)

- $r$ = 5600 employees; each attribute size 7 bytes, B = 64 bytes; P = 7 bytes; $n$ = NDV(Salary) = 280, salary is *uniformly* distributed.
- **B+ Tree of level *t*** on non-ordering, non-key Salary: leaf node fits in 1 block.

```
SQL2:SELECT * FROM EMPLOYEE
     WHERE Salary = 50000 OR Salary = 60000
```

**Task 4:** A disjunction query.

- B+ Tree is built over the Salary, thus, we *treat* this query as 2 different queries and then we *union* the results.

- Total cost: $t$ + 23 (for Salary = 50K) and $t$ + 23 (for Salary = 60K) thus, in total: **2*t*+46 block accesses.**

Search for Salary = 50K possibly follows different path w.r.t. search for Salary = 60K.

# EXAM QUESTION 3 (2019) 1/4

**EMPLOYEE**(<u>ID</u>, Name, Address, DNO)

Record size **R** = 256 bytes; *bfr* = floor(512/256) = 2 records/block.

Hash field: DNO with $\boldsymbol{y = h(\text{DNO}) = \text{DNO mod } 3}$ **(M = 3 buckets)**

**Task 1:** Design the structure of the M = 3 buckets that accommodates *r* = 7 tuples with DNO values: **0, 2, 3, 4, 6, 8, 9**.

Bucket M = 0: [0, 3] → [6, 9] (2 blocks)

Bucket M = 1: [4, _] (1 block)

Bucket M = 2: [2, 8] (1 block)

# Exam Question 3 (2019) 2/4

Bucket M = 0: [0, 3] → [6, 9] (2 blocks)

Bucket M = 1: [4, _] (1 block)

Bucket M = 2: [2, 8] (1 block)

**Task 2:** Calculate the *expected* cost in *best-/worst-case* scenario

```
SELECT * FROM EMPLOYEE WHERE DNO = x
```

Each bucket is selected with probability 1/3 = 0.33

- Bucket M = 0: 2 blocks in *worst-case* and 1 block in the *best-case*
- Bucket M = 1: 1 block in both cases
- Bucket M = 2: 1 block in both cases.

Worst-case: 0.33*2 + 0.33*1 + 0.33*1 = **1.32 block accesses**

Best-case: 0.33*1 + 0.33*1 + 0.33*1 = **1 block access**

# Exam Question 3 (2019) 3/4

Bucket M = 0: [0, 3] → [6, 9] (2 blocks)

Bucket M = 1: [4, _] (1 block)

Bucket M = 2: [2, 8] (1 block)

**Task 3:** Calculate the *expected* cost in *worst-case* scenario:

```
SELECT * FROM EMPLOYEE WHERE DNO >= 3 AND DNO <= 8
```

- Hashed File; Sequential File ordered by DNO; Heap File

- Hashing is not appropriate: issue a *new* query for each DNO in the range.
- There are 6 values in the range (DNO = 3, 4, 5 ,6, 7, and 8).
- **Total: 6 * 1.32 = 7.92 block accesses.**

# EXAM QUESTION 3 (2019) 4/4

```
SELECT * FROM EMPLOYEE WHERE DNO >= 3 AND DNO <= 8
```

- Sequential File has **4 blocks**:

  [0, 2], [3, 4], [6, 8], [9,_]

**Step 1:** $\log_2(4) = 2$ block accesses to locate the block with the DNO = 3

**Step 2:** Access the next contiguous block (up to DNO = 8).

**Total: 3 block accesses**.

- Heap File: *scan* the whole file, i.e., **4 block accesses.**

**Cost Ordering:** Sequential File < Heap File < Hash File

# Exam Question 4 (2019) 1/2

**EMPLOYEE**(SSN, Name, DNO) with **ordering non-key** DNO

DNO = 5 bytes; **R** = 100 bytes; $r$ = 1000 tuples, **B** = 256 bytes; pointer **P** = 5 bytes.

$n$ = 4 departments; employees are *uniformly* distributed over departments.

**Task:** Decide whether to build and use Clustering Index (DNO) or a serial scan:

```
SELECT * FROM EMPLOYEE WHERE DNO = x
```

- Blocking factor of the data file: $bfr$ = floor(256/100) = 2 records/block
- Index entry size: 5+5 = 10 bytes;
- blocking factor $m$ = floor(256/10) = 25 entries/block.
- File blocks $b$ = ceil(1000/2) = **500 blocks**; index blocks $ib$ = ceil(4/25) = **1 block.**

- **Overhead in storage:** (501-500)/500 = 1/500 or 0.2% additional storage.

# EXAM QUESTION 4 (2019) 2/2

**EMPLOYEE**(SSN, Name, DNO) with **ordering non-key** DNO

**Index Cost**: *All* employees are uniformly distributed over the departments

- Each cluster has on average 500/4 = **125 blocks**.
- 1 + 125 = **126 block accesses** for a DNO value.

**Baseline Cost:** *serial scan on sorted file with non-key attribute*

- DNO in {1,2,3,4}, each value with probability 1/4 = 0.25.
- If DNO = 1 then the access only the first 125 blocks.
- If DNO = 2 then we access the first and the next 125 blocks, i.e., 250 blocks
- If DNO = 3, then access the first 3 clusters of 125 blocks each, i.e., 375 blocks
- If DNO = 4 then we access the whole file, i.e., 500 blocks.

- **Expectation**: ¼*(125+250+375+500) = **312.5 block accesses.**
- Use the index!

# Exam Question 5 (2019) 1/4

EMPLOYEE(SSN, Salary, DNO);
$r$ = 10,000 tuples; $b$ = 2,000 blocks; blocking factor $f$ = 5

```
SELECT * FROM EMPLOYEE
WHERE    Salary >= 35000 AND DNO = 5
```

- **Clustering Index** on Salary *non-key ordering* with $x_{\text{Salary}} = 3$ levels.
- **B+ Tree** on DNO *non-key non-ordering* with $x_{\text{DNO}} = 2$ levels.
- 1 block with block-pointers per DNO value and NDV(DNO) = 125.
- Salary *in* [5,000 and 50,000].
- Available memory: 100 blocks.

**Task 1:** Estimate the *selection cardinality* of the query;

**Task 2:** Propose *two query processing plans* exploiting the access paths and select the *best* in terms of block accesses.

# Exam Question 5 (2019) 2/4

**Task 1:** *Conjunction* of two conditions, thus, obtain the *product* of selectivities:

*sel*(DNO)**sel*(Salary>=35000)

- $r$*(1/NDV(DNO) * ($max$(Salary)-35000)/($max$(Salary)-$min$(Salary))) = 26.4 tuples

# Exam Question 5 (2019) 3/4

**Task 2:** There are *two* possible plans *since* we have *two* conditions.

**Plan 1:** *execute* 'DNO = 5' and *then check* if 'Salary >= 35000'

- DNO is non-key, non-ordering (B+ Tree Index)
- **Target:** Retrieve employees working in Dept 5.
- Selection cardinality $s = r*sel$(DNO) = 80 employees in Dept. 5.
- 'Go-down the B+ Tree': 2 block accesses.
- 'Meet the block-pointers block': 1 block access.
- Each employee *may* be in a different block (*worst case*): access $s$ = 80 blocks.

- Cost-of-Plan 1: 2 + 1 + 80 = **83 block accesses.**
- Storage: 80 employees fit in ceil(80/5) = **16 blocks < available memory**.
- *In-memory*: check those employees with Salary >= 35K ☺

# EXAM QUESTION 5 (2019) 4/4

**Plan 2:** *execute* ‘Salary >= 35000’ and then check if ‘DNO = 5’

- Salary is non-key, ordering (Clustering Index)
- **Target:** Retrieve employees with Salary over £35K.
- ‘Go-down the Index’: 3 block accesses.
- Selection cardinality $s = r*sel$(Salary >= 35K) = 3300 employees with Salary >= 35K.
- ‘Meet all *contiguous* employees with over 35K’: 3300 employees fit in ceil(3300/5) = 660 contiguous blocks

- Cost-of-Plan 2: 3 + 660 = **663 block accesses.**
- Storage: 3300 employees (tuples) fit in **600 blocks**; **not available**.

**Winner is: Plan 1**

# EXAM QUESTION 7 (2018) (1/6)

**EMPLOYEE E & DEPARTMENT D**

- Relation E has $n_E$ = 100 blocks and $r_E$ = 1000 employees
- Relation D has $n_D$ = 50 blocks and $r_D$ = 10 department.
- Memory: $n_B$ = 12 blocks for processing; bfrRS = 10 matching-records/block.

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

- **Task 1:** Based on the **nested-loop join strategy**, estimate the expected cost and report on the **best** nested-loop strategy.
- **Task 2:** Estimate the expected cost of the *naïve* join strategy (Cartesian).
- **Task 3:** Secondary Index on MGR_SSN level $x_D$ = 2 and a Primary Index on SSN level $x_E$ = 2.
  - Propose **two index-based nested-loop join** strategies
- **Task 4:** Relations are sorted by SSN and MGR_SSN;
  - Which is the cost of the **sort-merge join strategy**?

# Exam Question 7 (2018) (2/6)

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

**Task 1: nested-loop join strategy;** report on the **best** strategy.

Join Selectivity: *js* = 1/max(NDV(SSN), NDV(MGR_SSN)) with: NDV(SSN) = 1000 and NDV(MGR_SSN) = 10, ***js* = 1/max(1000,10) = 1/1000 = 0.001.**

- Outer loop involves the relation with the *smallest* number of blocks: D

$n_D + n_E * \text{ceil}(n_D/(n_B\text{-}2)) + (js * rE * rD)/bfrRS$ = **551 block accesses.**

- **Note:** If E was used in the outer loop, then we would have obtained:

$n_E + n_D * \text{ceil}(n_E/(n_B\text{-}2)) + (js * rE * rD)/bfrRS$ = **601 block accesses.**



Best strategy: use relation D at the outer loop with cost: **551 block accesses**

# Exam Question 7 (2018) (3/6)

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

**Task 2:** Expected cost of the naïve join (producing the Cartesian product).

Based on the naïve join:

- $n_E * n_D + (js * rE * rD)/bfrRS$ = **5,001 block accesses**.

# Exam Question 7 (2018) (4/6)

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

**Task 3:** Secondary Index on MGR_SSN level $x_D = 2$ and a Primary Index on SSN level $x_E = 2$.

Propose **two index-based nested-loop join** strategies

**[Strategy 1]** Retrieve each tuple *e* from E and use the Index (MGR_SSN) to find the *matching* tuple *d* in D;

**Recall:** *not* all employees are managers!

**Cost 1:** $n_E + r_E * (x_D + 1) + (js * rE * rD)/bfrRS$ = **3,101 block accesses;**

# EXAM QUESTION 7 (2018) (5/6)

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

**Task 3:** Secondary Index on MGR_SSN level $x_D = 2$ and a Primary Index on SSN level $x_E = 2$.

Propose **two index-based nested-loop join** strategies

**[Strategy 2]** Retrieve each tuple *d* from D and use the Index (SSN) to find the *matching* tuple *e* in E;

**Recall:** *every* department D has a matching record in E.

**Cost 2**: $n_D + r_D*(x_E + 1) + (js * rE * rD)/bfrRS$ = **81 block accesses;**

**Best: Strategy 2**

# EXAM QUESTION 7 (2018) (6/6)

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

**Task 4:** Employee and Department are sorted by SSN and MGR_SSN;
Which is the join cost of the **sort-merge join strategy**?

Both relations are sorted with their join attributes:

- **Cost**: $n_E + n_D + (js * r_E * r_D)/bfrRS$ = **151 block accesses.**