Databázové systémy 2

Michal Krátký

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – Technická univerzita Ostrava

2024/2025

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

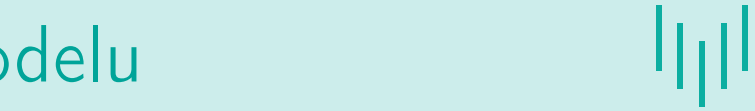
Obsah přednášky

Motivace

```
create table Customer (
  idCustomer int primary key,
  fName varchar(20) not null,
  lName varchar(30) not null,
  residence varchar(20) not null,
  gender char(1) not null,
  birthday date not null
);
insert into Customer values(1, 'Jan', 'Sobota', ...)
```

- Do jakých datových struktur jsou ukládány záznamy?
- Jaké vlastnosti/složitosti základních operací tyto datové struktury poskytují?
- Můžeme zrychlit provádění SQL dotazů?

Implementace relačního datového modelu



- Základní úložiště pro tabulku relačního datového modelu je **tabulka typu halda (heap table)**:
  - stránkovaný propojený seznam.
- **Stránky/bloky:**
  - Záznamy jsou uloženy ve stránkách/blocích o velikosti nejčastěji 8kB (násobky alokačních jednotek souborového systému - nejčastěji 2kB).
  - Stránky slouží pro efektivní výměnu dat mezi pamětí (*cache buffer*) a diskem.
- **Vyhledávání je sekvenční:** DBS prochází všechny stránky a všechny záznamy.
  - Složitost $O(n)$, kde n je počet záznamů.
  - IO cost = N, kde N je počet bloků haldy.

Počet bloků haldy - Oracle

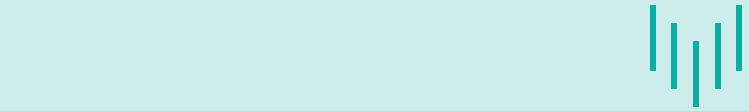
- Celkový počet bloků zjistíme například z pohledu systémové katalogu user_segments:

```
select count(*) from Customer;
select blocks from user_segments
  where segment_name = 'CUSTOMER';

COUNT(*)
----------
300000

BLOCKS
----------
2048
```

Počet bloků tabulky, SQL Server



```
SELECT
  t.NAME AS TableName,
  p.rows AS RowCounts,
  SUM(a.total_pages) AS TotalPages,
  SUM(a.used_pages) AS UsedPages
FROM sys.tables t
INNER JOIN
  sys.indexes i ON t.OBJECT_ID = i.object_id
INNER JOIN
  sys.partitions p ON i.object_id = p.OBJECT_ID AND
    i.index_id = p.index_id
INNER JOIN
  sys.allocation_units a ON p.partition_id = a.container_id
WHERE t.NAME = 'Customer'
GROUP BY t.Name, p.Rows
ORDER BY t.Name
```

```
TableName   RowCounts   TotalPages  UsedPages
Customer    300000      2426        2424
```

Mazání záznamů 1/3

- Záznamy v tabulce nejsou nijak uspořádány: mazání po každé operaci `delete`, by v nejhorším případě, znamenalo přesouvání n záznamů v hladě.
- Záznamy proto nejsou **fyzicky mazány**, jsou pouze **označeny jako smazané**.
- Složitost je tedy teoreticky $O(1)$, ale ...
  - Operaci mazání často předchází operace vyhledání, prakticky je tedy složitost v $O(n)$.
  - Podobně DBS musí kontrolovat případné **cizí klíče** v ostatních tabulkách, zda není mazán záznam, na který se jiný záznam odkazuje (kontrola **referenční integrity**).
- **Důsledek ne-mazání z hlady:** počet bloků haldy se po operaci mazání nesnižuje.

Mazání záznamů 2/3

```
create table OrderItems (
  idOrder int references "Order"(idOrder) not null,
  idProduct int references Product(idProduct) not null,
  unit_price int not null,
  quantity int not null,
  primary key(idOrder, idProduct));

select count(*) from OrderItems;
select blocks from user_segments
  where segment_name = 'ORDERITEMS'; -- Oracle
```

```
COUNT(*)
----------
1000000

BLOCKS
----------
3328
```

Mazání záznamů 3/3

```
delete from OrderItems;

1 000 000 rows deleted.

select count(*) from OrderItems;
select blocks from user_segments
  where segment_name = 'ORDERITEMS';
```

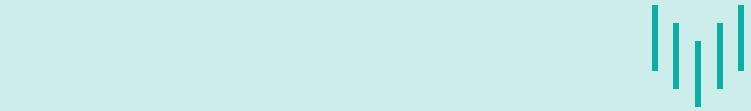
COUNT(*)	BLOCKS
0	3328

Záznamy byly smazány, počet bloků je ale stejný: bloky jsou prázdné.
DBS nabízí operaci pro fyzické smazání záznamů označených ke smazání (`alter table rebuild` pro SQL Server, `shrink` pro Oracle).

Vkládání záznamů

- Při **vkládání** je záznam umístěn na první nalezenou volnou pozici ve stránce haldy (časová složitost $O(n)$) nebo do poslední stránky (složitost $O(1)$).
- V ideálním případě se **využití stránek** (**utilization**) blíží 100%.
  - Bloky haldy jsou tedy plné.
- Teoretická složitost vkládání je $O(1)$, ale ...
  - Pro **primární klíče** a **jedinečné atributy** je nutné kontrolovat jedinečnost hodnot – v haldě má tato kontrola složitost $O(n)$.
  - Podobně DBS musí kontrolovat hodnoty pro **cizí klíče**, zda se záznam nachází v odkazované tabulce (kontrola **referenční integrity**).

Plán vykonávání dotazu



- Pro databázovou operaci (zjednodušeně SQL dotaz) DBS (optimalizátor) počítá několik **plánů vykonání dotazu** (angl. **query execution plan - QEP**).
- DBS odhadne, který plán je nejlepší (nejrychlejší) a dle tohoto plánu dotaz provede.
- V DBS máme možnost zobrazit vybraný **plán vykonávání dotazu**, který obsahuje provedené **logické operace** (operace relační algebry) i **fyzické operace** (konkrétní operace konkrétních datových struktur).
  - Tento plán může sloužit pro ladění dotazu.
  - Zobrazení plánu je závislé na DBS.

Cena operací plánu

Cenu operací měříme pomocí:

- **IO cost** – počet přístupů ke stránkám datových struktur.
- **CPU cost** – počet operací, např. počet porovnání provedený při provádění operace.
- **Času provedení operace (Processing time)** – je relativní, používáme je při ladění operací na jednom počítači.

Zobrazení plánu vykonání dotazu, Oracle 1/2

- Plán dotazu se příkazem `explain plan` uloží do tabulky `PLAN_TABLE`

  Například:

```
explain plan for
select * from Customer
  where birthday = TO_DATE('01.01.2000',
    'DD.MM.YYYY');
```

- Záznamy v tabulce `PLAN_TABLE` nejsou automaticky mazány, musíme je tedy mazat ručně pomocí `DELETE PLAN_TABLE`

Zobrazení plánu vykonání dotazu, Oracle 2/2

- Pro zobrazení plánu použijeme dotaz nad `PLAN_TABLE` nebo můžeme využit následující funkci:

```
select * from table(dbms_xplan.display);
```

- Plán je reprezentován jako strom: operace plánu jsou vykonávány od listů směrem ke kořeni.
- Plán pro dotaz (výsledek obsahuje 8 záznamů) je následující[1]:

```
Operation                      Object                Cost      CPU Cost
------------------------------ --------------------- --------- ----------
SELECT STATEMENT ()                                  553       111238533
  TABLE ACCESS (FULL)          CUSTOMER              553       111238533
```

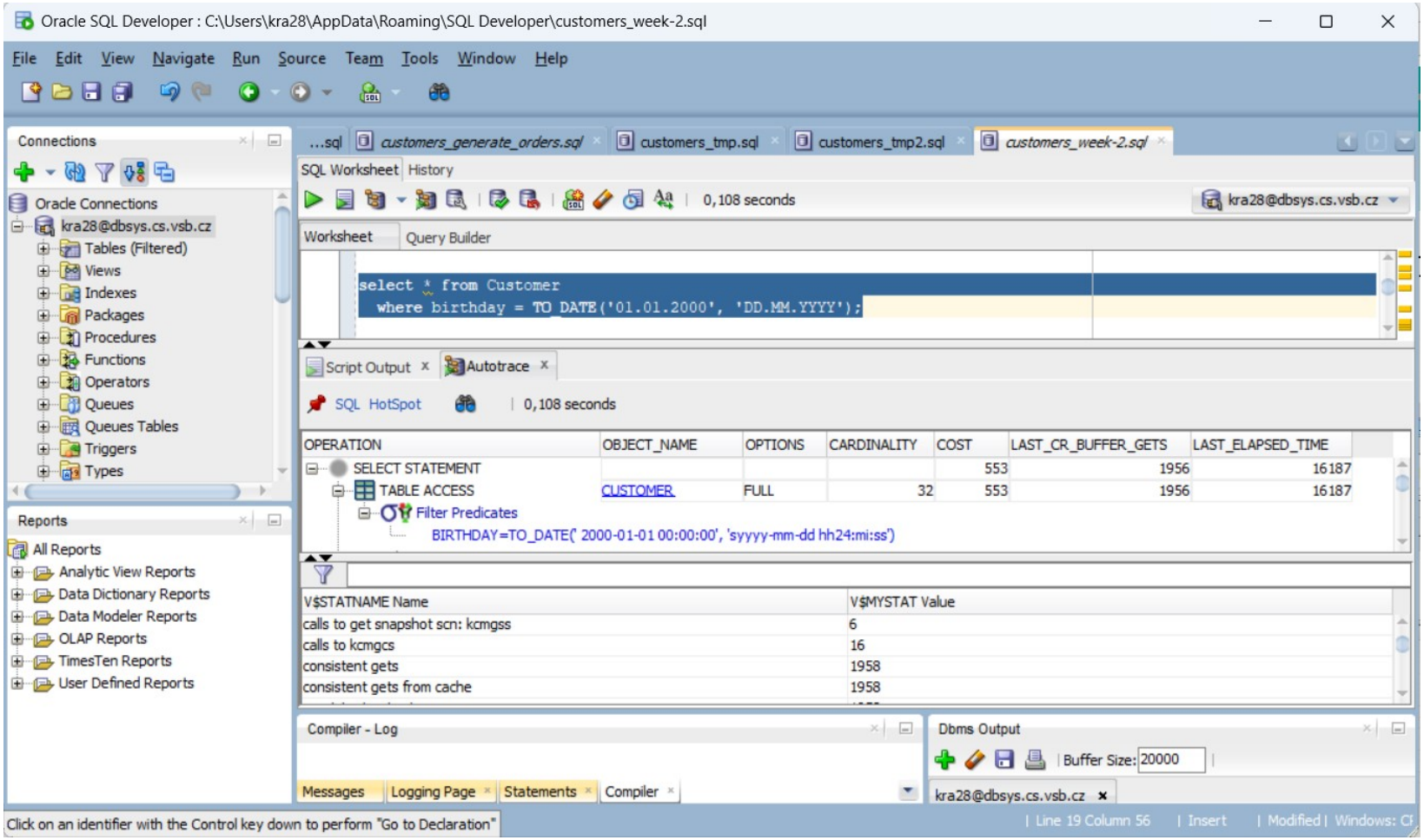
- Máme tedy první fyzickou operaci[2]: **TABLE ACCESS (FULL)** – **sekvenční průchod** všemi bloky haldy.

[1] Cost není IO cost.

[2] Názvosloví operací je závislé na konkrétním DBS.

Grafické zobrazení plánu v Oracle, Autotrace

V Oracle získáme podrobnější informace pomocí příkazu `SET AUTOTRACE ON`, nebo využijeme Autotrace v SQL Developer označením příkazu a stisknutím F6.



Oracle, IO cost, buffer gets[3] 1/5

IO cost se v Oracle skrývá pod poznačením **buffer gets**, hodnotu získáme z pohledu systémového katalogu `v$sql`

```
select executions as executions,
  buffer_gets/executions as buffer_gets,
  (cpu_time/executions)/1000.0 as cpu_time_ms,
  (elapsed_time/executions)/1000.0 as elapsed_time_ms,
  rows_processed/executions as rows_processed, parsing_user_id,
  sql_id, plan_hash_value,sql_text from v$sql
where sql_id='g4mshxbky6215' and plan_hash_value=2844954298;
```

```
EXEC. BUFFER_GETS CPU_TIME_MS ELAPSED_TIME_MS ROWS_PROCESSED PARSING_USER_ID SQL_ID        PLAN_HASH_VALUE
----- ----------- ----------- --------------- -------------- --------------- ------------- ---------------
19    1955.31579  15.9395789  15.2995789      75             107             g4mshxbky6215 2844954298

SQL_TEXT
--------------------------------------------------
select * from Customer where lname = 'Svobodová' and fname='Jana' and residence = 'Ostrava'
```

[3] https://docs.oracle.com/cd/B14117_01/server.101/b10755/dynviews_2097.htm

buffer gets 2/5

```
EXEC. BUFFER_GETS CPU_TIME_MS ELAPSED_TIME_MS ROWS_PROCESSED PARSING_USER_ID SQL_ID        PLAN_HASH_VALUE SQL
----- ----------- ----------- --------------- -------------- --------------- ------------- --------------- ---
19    1955.31579  15.9395789  15.2995789      75             107             g4mshxbky6215 2844954298      sel
```

Význam atributů:

- `executions` (EXEC.): počet provedení dotazu. Některé hodnoty jsou součtem pro všechna provedení, proto jsou tyto hodnoty děleny hodnotou `executions`.
- `buffer_gets`: počet logických přístupů. Počet bloků haldy je v tomto případě 2 048, Oracle reportuje IO cost = 1955.
- `parsing_user_id`: id uživatele, můžeme získat tímto dotazem:

```
SELECT user_id FROM dba_users WHERE username='KRA28';
```

buffer gets 3/5

EXEC.	BUFFER_GETS	CPU_TIME_MS	ELAPSED_TIME_MS	ROWS_PROCESSED	PARSING_USER_ID	SQL_ID	PLAN_HASH_VALUE	SQL
19	1955.31579	15.9395789	15.2995789	75	107	g4mshxbky6215	2844954298	sel

Význam atributů:

- `cpu_time_ms`: čas provedení dotazu v ms, v DB je uložen v μs.
- `elapsed_times_ms`: elapsed time = cpu time + user i/o wait time + application wait time + ...[4] v ms, v DB je uložen v μs.
- `rows_processed`: počet záznamů výsledku.

[4] `https://www.dba-oracle.com/m_sql_execute_elapsed_time.htm`

buffer gets 4/5

```
EXEC. BUFFER_GETS CPU_TIME_MS ELAPSED_TIME_MS ROWS_PROCESSED PARSING_USER_ID SQL_ID        PLAN_HASH_VALUE SQL
----- ----------- ----------- -------------- -------------- --------------- ------------- --------------- ---
19    1955.31579  15.9395789  15.2995789     75             107             g4mshxbky6215 2844954298      sel
```

Význam atributů:

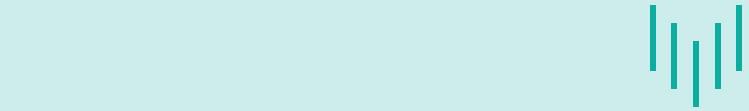
- `sql_id`: id SQL dotazu, můžeme získat například takto:

```
set feedback on SQL_ID;
select * from Customer where lname = 'Svobodová' and fname='Jana' and
residence = 'Ostrava';
set feedback off SQL_ID;

298049 Jana     Svobodová        Ostrava              f 30-AUG-76
...
75 rows selected.
SQL_ID: g4mshxbky6215
```

Pozor: dotaz musí být spouštěn označením a F5 (Run Script) pro získání kompletního výsledku dotazu. Pokud budete spouštět Ctrl+Enter (Run Statement), bude docházet ke stránkování výsledku na počet záznamů dle `Sql Array Fetch Size`.

buffer gets 5/5



```
EXEC. BUFFER_GETS CPU_TIME_MS ELAPSED_TIME_MS ROWS_PROCESSED PARSING_USER_ID SQL_ID        PLAN_HASH_VALUE SQL
----- ----------- ----------- -------------- -------------- --------------- ------------- --------------- ---
19    1955.31579  15.9395789  15.2995789     75             107             g4mshxbky6215 2844954298      sel
```

Význam atributů:

- plan_hash_value: číslo plánu, můžeme získat například takto:

```
explain plan for select * from Customer where lname = 'Svobodová' and
fname='Jana' and residence = 'Ostrava';

select * from table(dbms_xplan.display);

Plan hash value: 2844954298

-------------------------------------------------------------------------------
| Id | Operation          | Name     | Rows  | Bytes | Cost(%CPU)| Time     |
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT   |          |  2308 | 94628 |   552 (2) | 00:00:01 |
|  1 |  TABLE ACCESS FULL| CUSTOMER |  2308 | 94628 |   552 (2) | 00:00:01 |
-------------------------------------------------------------------------------
```

Zobrazení plánu vykonání dotazu, SQL Server

- Textové zobrazení plánu vykonávání dotazu:
  `SET SHOWPLAN_TEXT ON`
- Můžeme přečíst pořadí operací: plán se zobrazí jako strom, operace jsou vykonávány od listů.
- Pro dotaz[5]:

```
select * from Customer
where birthday =  '2000-01-01';
```

- Dostaneme plán:

```
|--Table Scan(OBJECT:([kra28].[dbo].[Customer]),
  WHERE:([kra28].[dbo].[Customer].[birthday]='2000-01-01'))
```

- Sekvenční průchod haldou se tedy v SQL Server nazývá **Table Scan**, zatímco v Oracle **Table Access (Full)**.

[5] Výsledek obsahuje 21 záznamů.

SQL Server, IO cost 1/2

- V SQL Server můžeme IO cost získat následujícím nastavením:

```
SET STATISTICS IO ON;
```

- Pro dotaz:

```
select * from Customer
where birthday =  '2000-01-01';
```

- Získáme tento výstup:

```
(21 rows affected)
Table 'Customer'. Scan count 1, logical reads 1750,
physical reads 0, ...
```

- IO cost = 1 750.

SQL Server, IO cost 2/2

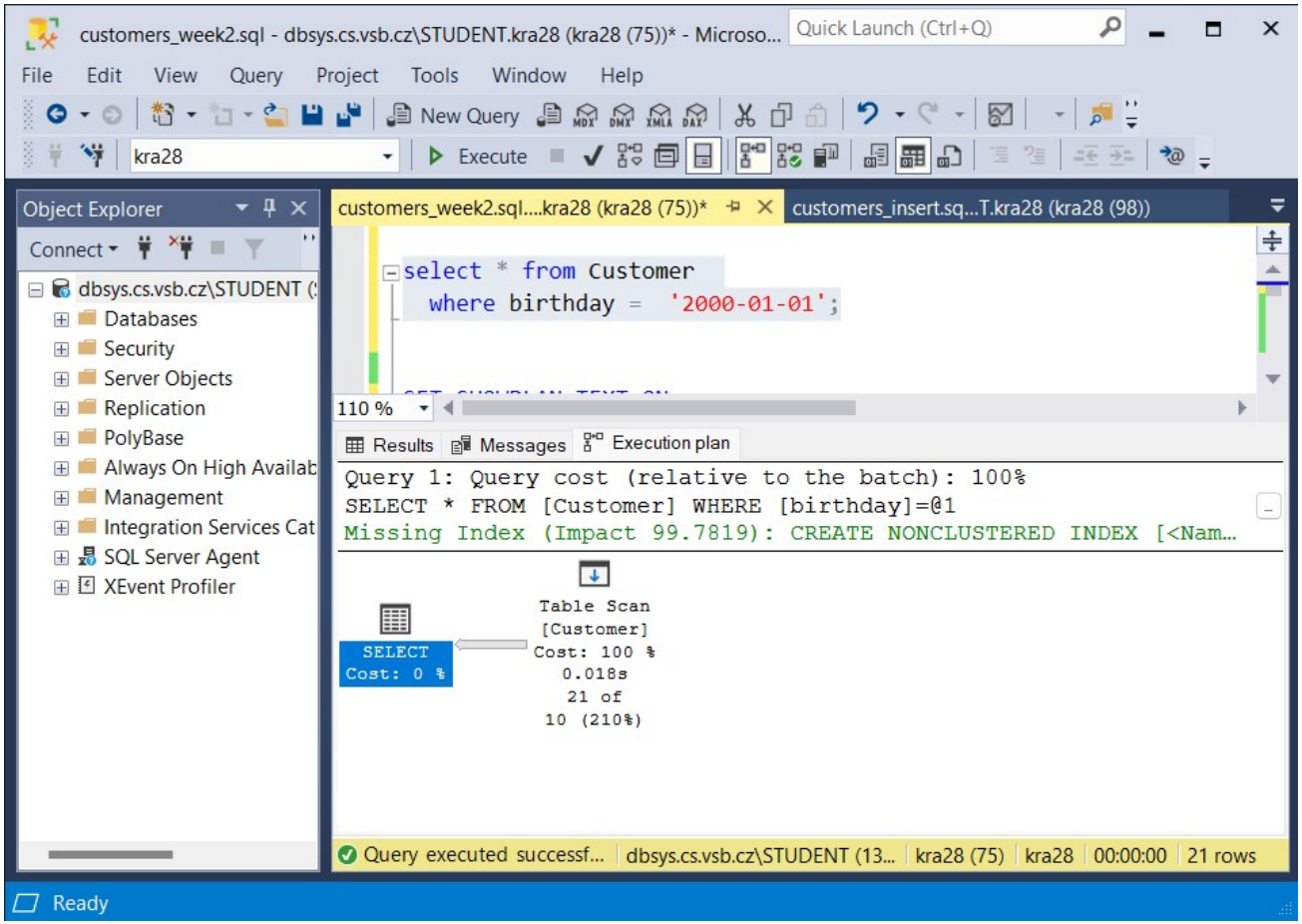
- Výstup:

```
(21 rows affected)
Table 'Customer'. Scan count 1, logical reads 1750,
physical reads 0, ...
```

- Přístupy ke stránkám dělíme na:
  - **logical reads – logické přístupy** (buffer gets v Oracle).
  - **physical reads – fyzické přístupy** (physical reads v Oracle): stránky nejsou v cache buffer a musí být načteny z disku.
- Pokud se nám, i při opakování dotazu, stále objevují nenulové fyzické přístupy, musíme zvětšit cache buffer.

Grafické zobrazení plánu dotazu, SQL Server

Management Studio, menu: Query/Include Actual Execution Plan



V grafickém plánu vidíme relativní podíl operace, v tomto případě 100% pro **Table Scan**, na režii vykonání dotazu.

Měření času provedení dotazu v SQL Server

- Pro zobrazení času provedení dotazu na serveru použijeme:

```
SET STATISTICS TIME ON;

select * from Product
where unit_price between 1300000 and 1800000;

SQL Server parse and compile time:
CPU time = 3 ms, elapsed time = 3 ms.

(1958 rows affected)

SQL Server Execution Times:
CPU time = 16 ms,  elapsed time = 51 ms.
```

  - `SQL Server parse and compile time` zahrnuje parsování a zpracování dotazu, generování plánů atd. Při dalších spuštění dotazu bude pravděpodobně 0.
  - `SQL Server Execution Times` měří samotný čas provedení dotazu. `elapsed time` je `CPU time` + další režie (např. zaslání výsledku). **Většinou budeme používat** `CPU time`**.**

Sekvenční průchod haldou

- Sekvenční průchod se v tomto případě provádí:
  - **Při dotazech s nízkou selektivitou**, např. dotazech vracejících všechny záznamy tabulky:

```
select * from Customer;
```

  - **Tak pro dotazy s vysokou selektivitou**, např. pro tento dotaz vracející jednotky záznamů:

```
select * from Customer
where birthday =  '2000-01-01';
```

- Při ladění dotazu se budeme zaměřovat především na dotazy s **vysokou selektivitou** jejichž QEP obsahují sekvenční průchod haldou.

Index pro primární klíč, Oracle

- Uvažujme nyní dotaz na hodnotu primárního klíče, například:

```
select * from Customer where idCustomer=100001;
```

- Plán v Oracle:

```
Operation                                Object
---------------------------------------- ----------------------
SELECT STATEMENT ()
  TABLE ACCESS (BY INDEX ROWID)          CUSTOMER
    INDEX (UNIQUE SCAN)                  SYS_C00469561
```

- buffer gets je **3 namísto 1 958** jako u dotazu, který prováděl sekvenční průchod haldou.
- Ačkoli jsme žádný index nevytvořili pomocí create index, DBS **automaticky vytvořil index pro primární klíč**.

Index pro primární klíč, SQL Server

- Uvažujme stejný dotaz na hodnotu primárního klíče:

```
select * from Customer where idCustomer=100001;
```

- Plán v SQL Server:

```
|--Nested Loops(Inner Join, ...)
  |--Index Seek(OBJECT:...
       [PK__Customer__D05876871149C525]), SEEK: ...)
  |--RID Lookup(OBJECT:(....[Customer]), SEEK: ...)
```

- `logical reads` je **4 namísto 1 750** jako u dotazu, který prováděl sekvenční průchod haldou.
- Opět vidíme automatické vytvoření indexu na primární klíč.

Zjištění indexů vytvořených pro tabulku, Oracle

- V Oracle využijeme pohledu systémového katalogu `user_indexes`.

```
select index_name from user_indexes
where table_name='CUSTOMER';

INDEX_NAME
--------------------
SYS_C00469561
```

Zjištění indexů vytvořených pro tabulku, SQL Server

- V SQL Server využijeme pohledů systémového katalogu `sys.tables` a `sys.indexes`.

```
SELECT
   TableName = t.Name,
   i.*
FROM sys.indexes i
INNER JOIN
   sys.tables t ON t.object_id = i.object_id
WHERE
   T.Name = 'Customer' and i.name is not null;
```

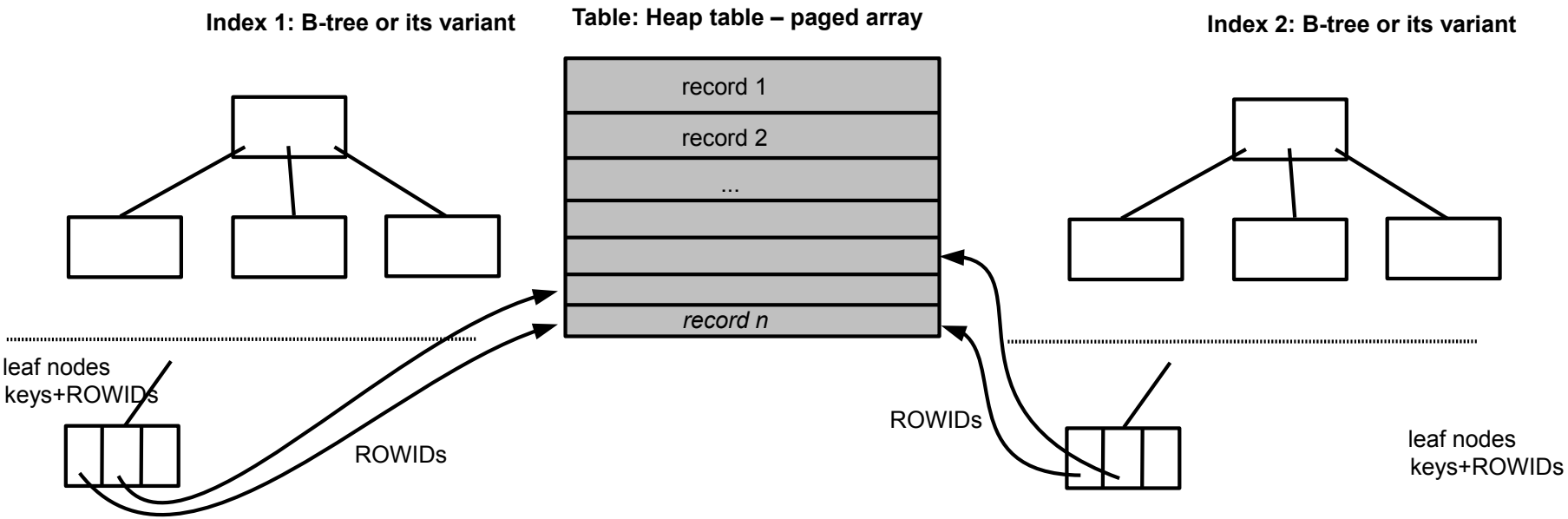
```
TableName   object_id   name ...
Customer    2133582639  PK__Customer__D05876871149C525 ...
```

Proč je automaticky vytvořen index pro primární klíč?

- Index je datová struktura **B-strom**, který poskytuje **logaritmické časové složitosti** základních operací (kromě rozsahového dotazu).
- Pro bodový dotaz, tedy dotaz na jednu hodnotu, např. hodnotu primárního klíče, je tedy výrazně rychlejší než halda s lineární složitostí vyhledávání.
- Proč je index automaticky vytvářen?
  - Rychlejší kontrola jedinečnosti primárního klíče při operaci `insert`.
  - Rychlejší provedení bodových dotazů na primární klíč.
  - Rychlejší kontrola referenční integrity, když se pokoušíme smazat záznam s primárním klíčem, na který se může odkazovat záznam s cizím klíčem z jiné tabulky.

Základní úložiště pro relační datový model

- Ke každé tabulce je vytvořena halda a index typu B-strom pro primární klíč a jedinečné atributy (`unique`).
- Index obsahuje v položkách stránek dvojice (hodnota primárního klíče, **ROWID**), kde ROWID (v SQL Server **RID**) je odkaz na záznam do haldy.



ROWID

- Pomocí **ROWID** identifikujeme záznamy v haldě.
- ROWID je 8-10B hodnota (v Oracle 10B, v SQL Server 8B) skládající se z:
  - čísla bloku,
  - pozice záznamu v haldě.
- Proč DBS nepoužívá paměťový ukazatel?
  - protože bloky mohou být umístěny na disku.

Vykonávání dotazů na hodnotu primárního klíče

- Dotazy se selekcí na primární klíč, vykonávané pomocí indexu, pak pracují v těchto krocích:
  1. Pomocí **bodového dotazu v indexu typu B-strom** je nalezena hodnota atributu a s ním ROWID/RID příslušného záznamu.
     - Oracle: `INDEX (UNIQUE SCAN)`
     - SQL Server: `Index Seek`
  2. Pomocí ROWID/RID, DBS přímo získá záznam z haldy:
     - Oracle: `TABLE ACCESS (BY INDEX ROWID)`
     - SQL Server: `RID Lookup`

Vykonávání dotazů pomocí indexu

- IO cost:
  1. **Bodový dotaz v indexu typu B-strom**: IO cost $= h + 1$, kde h je výška stromu, $\lceil h = \log n \rceil$.
     - Pokud se počet položek stránky B-stromu pohybuje ve stovkách, je h, i pro velký počet položek, v jednotkách, např. 3 – 4.
  2. **Získání záznamu pomocí ROWID**:
     - Přímý přístup k bloku v cache buffer: IO cost $= 1$.
- IO cost dotazu se selekcí na primární klíč, vykonaný pomocí indexu, je tedy: **IO cost** $= h + 2$.

IO cost dotazu se selekcí na hodnotu primárního klíče

- SQL dotaz:

```
select * from Customer where idCustomer=100001;
```

- **QEP, Oracle:**

```
Operation                                 Object
----------------------------------------  ----------------------
SELECT STATEMENT ()
  TABLE ACCESS (BY INDEX ROWID)           CUSTOMER
    INDEX (UNIQUE SCAN)                   SYS_C00469561
```

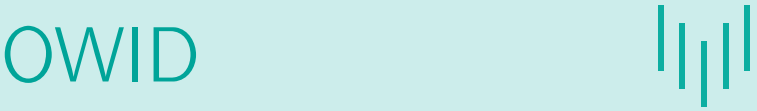
  - buffer gets = 3

- **QEP, SQL Server:**

```
|--Nested Loops(Inner Join, ...)
   |--Index Seek(OBJECT:...
      [PK__Customer__D05876871149C525]), SEEK: ...)
   |--RID Lookup(OBJECT:(....[Customer]), SEEK: ...)
```

  - logical reads = 4

Bodový dotaz na index bez využití ROWID



- Je možné napsat dotaz na hodnotu primárního klíče, který nevyužije ROWID pro získání záznamu?
- Ano je, stačí nepoužít projekci na ostatní atributy:

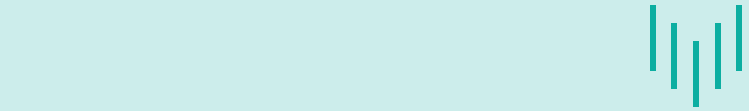
```
select count(*) from Customer
where idCustomer=100001;
```

- Plán v SQL Server:

```
|--Index Seek(OBJECT:(... [PK__Customer__D05876871149C525]
SEEK: ...)
```

- `logical reads` je 3, namísto 4, kdy jsme použili `select *`.

Index typu B-strom



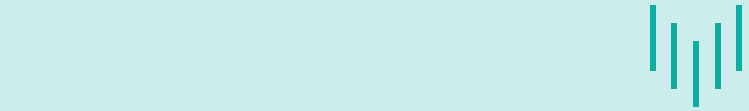
- V případě vysoce selektivních dotazů, tj. dotazů vracející relativně málo záznamů vůči počtu záznamů tabulky, je vhodné vytvořit index příkazem:
  `CREATE INDEX <index_name> ON <table_name>(<list of attributes>)`
- Ve většině databázových systémů se vytvoří datová struktura B-strom, zveřejněná v článku:

  Rudolf Bayer, Edward M. McCreight: **Organization and Maintenance of Large Ordered Indices**. Acta Informatica 1: 173-189 (1972).
- Nejčastěji se používá varianta B^+-strom, která obsahuje indexované položky pouze v listových uzlech.

B^+-strom

- Základní vlastnosti B^+-stromu řádu C:
  - Snadno stránkovatelný (srovnejme s binárním stromem): C nastavíme dle velikosti stránky např. 8kB.
  - Vyvážený: vzdálenost od všech listů ke kořenovému uzlu je stejná.
  - Výška stromu $h \approx \lceil \log_C(n) \rceil \Rightarrow$ maximální počet záznamů $N = C^h$
  - Mazání, vkládání a bodový dotaz mají **časovou složitost** $O(\log_C(n))$
  - Výška je vzdálenost od kořene k listu (počet hran), IO cost bodového dotazu je tedy $h + 1$.

Velikost indexu



- Kvůli štěpení uzlů, je **využití uzlů** (**node utilization**) $\geq 50\%$ (u haldy, pokud neuvažujeme operaci delete, je to 100%).
  - $\Rightarrow$ ačkoli je položka B-stromu menší než záznam (obsahuje jeden nebo několik málo atributů a ROWID), index je relativně větší než halda.
- Oracle:

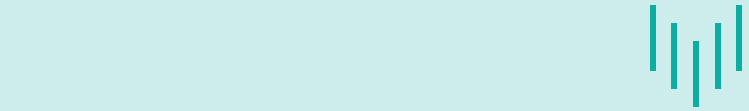
```
select blocks from user_segments
where segment_name = 'CUSTOMER';

select blocks from user_segments
where segment_name = 'SYS_C00470718';

BLOCKS
----------
2048
640
```

- Klíč zahrnuje jen jeden atribut ze 6 atributů tabulky `Customer`, ale velikost indexu je 31% velikosti haldy.

Počet bloků indexu, SQL Server



```
create or alter procedure printIndexPages
@indexName varchar(30)
AS
  SELECT
    i.name AS IndexName,
    p.rows AS ItemCounts,
    SUM(a.total_pages) AS TotalPages,
    SUM(a.used_pages) AS UsedPages
  FROM sys.indexes i
  INNER JOIN
    sys.partitions p ON i.object_id = p.OBJECT_ID AND
      i.index_id = p.index_id
  INNER JOIN
    sys.allocation_units a ON p.partition_id = a.container_id
  WHERE i.name = @indexName
  GROUP BY i.name, p.Rows
  ORDER BY i.name
```

Velikost indexu, SQL Server

- SQL Server:

```
exec printTablePages 'Customer';
exec printIndexPages 'PK__Customer__D0587687CA5DC7CF
```

```
TableName   RowCounts   TotalPages  UsedPages
Customer    300000      2426        2424
```

```
IndexName                          ItemCounts  TotalPages  UsedPages
PK__Customer__D0587687CA5DC7CF     300000      673         673
```

- Klíč zahrnuje jen jeden atribut ze 6 atributů tabulky Customer, ale velikost indexu je 28% velikosti haldy.

Rozsahový dotaz v B-stromu

- **Rozsahový dotaz v B-stromu** je vykonán pokud DBS očekává víc než jednu položku ve výsledku.
- Rozsahový dotaz v B-stromu je typicky vykonán pro rozsahový dotaz v SQL, např.:

```
create index product_unit_price on Product(unit_price

select * from Product
where unit_price between 20793000 and 20796000;
```

- Rozsahový dotaz v B-stromu **je vykonán** takto:
  1. Bodový dotaz pro nižší hodnotu rozsahu, v tomto případě 20 793 000.
  2. Poté jsou porovnávány další klíče v bloku, dokud klíč $<=$ vyšší hodnota rozsahu, v tomto případě 20 796 000.
  3. Pokud je porovnán kompletní blok, je načtena následující listová stránka (každá listová stránka B^+-stromu obsahuje odkaz na následující listovou stránku).

Rozsahový dotaz - rozsahový SQL dotaz 1/2

- Uvažujme tento rozsahový SQL dotaz:

```
select * from Product
where unit_price between 20793000 and 20796000;
```

- Velikost výsledku:
  - Oracle: 5
  - SQL Server: 4

Rozsahový dotaz - rozsahový SQL dotaz 2/2

- **Plán vykonání dotazu:**
  - Oracle:

```
SELECT STATEMENT
  TABLE ACCESS (BY INDEX ROWID)   PRODUCT
    INDEX (RANGE SCAN)            PRODUCT_UNIT_PRICE
```

  - SQL Server:

```
|--Nested Loops(...)
   |--Index Seek(OBJECT:(...[product_unit_price]),...)
   |--RID Lookup(OBJECT:(...[Product]), ...)
```

    - SQL server nerozlišuje bodový/rozsahový dotaz.

Složený klíč indexu

- Pokud klíč obsahuje více než jeden atribut $a_1, a_2, ...a_k$, mluvíme o **složeném klíči**.
- **Jaké dotazy** umožní B-strom efektivně vykonat?
- **Záleží na pořadí** atributů v definici `create index`?

Tabulka OrderItems

IDORDER	IDPRODUCT	UNIT_PRICE	QUANTITY
1	4320	1796023	1
1	7795	28533	9
1	24477	4157	9
1	25231	41566	6
1	34709	1497974067	1
2	19090	62625	8
2	24733	71542	10
2	42795	61306	5
2	48281	95708	10
2	51968	31302	5
2	86756	2274	10
3	35455	1710186106	2
3	36256	2928345	1
3	44360	22510816	3
3	58269	27514	7
3	75299	70638	2
3	81503	35979	6

...

- Klíče jsou v listových uzlech B-stromu setřízeny dle atributů $a_1, a_2, ...a_k$, tak jak byly uvedeny v definici `create index`.
- Mluvíme o tzv. **lexicografickém uspořádání** (viz třízení slov ve slovníku).
- V tomto případě jsou tedy nejprve uloženy klíče s `idOrder=1`, setřízené dle `idProduct`, pak klíče s `idOrder=2` atd.

Složený klíč – bodový dotaz

- Aby DBS mohl využít index, musí dotaz odpovídat lexikografickému uspořádání, tj. pořadí atributů v klíči:

```
select * from OrderItems
where idOrder=12456 and idProduct=47506; -- 1 zaznam
```

- **QEP:** (idOrder, idProduct) je primární klíč, bude proveden bodový dotaz v B-stromu.
  - Oracle:

```
SELECT STATEMENT
  TABLE ACCESS BY INDEX ROWID  ORDERITEMS
    INDEX UNIQUE SCAN          SYS_C00470736
```

  - SQL Server:

```
|--Nested Loops(...)
   |--Index Seek(OBJECT:(...[PK__OrderIte__CD443163A0C75C37])
   |--RID Lookup(OBJECT:(...[OrderItems]), ...)
```

- **IO cost:** buffer gets = 4, logical reads = 4.

Složený klíč – rozsahový dotaz

- Aby DBS mohl využít index, musí dotaz odpovídat lexikografickému uspořádání:

```
select * from OrderItems
where idOrder=12456; -- 16 resp. 17 zaznamu
```

- **QEP**: (k jedné hodnotě `idOrder` náleží 0 a více záznamů, bude proveden rozsahový dotaz v B-stromu).
  - Oracle:

```
SELECT STATEMENT
  TABLE ACCESS BY INDEX ROWID BATCHED    ORDERITEMS
    INDEX RANGE SCAN                     SYS_C00470736
```

  - SQL Server:

```
|--Nested Loops(...)
   |--Index Seek(OBJECT:(...[PK__OrderIte__CD443163A0C75C37])
   |--RID Lookup(OBJECT:(...[OrderItems]), ...)
```

- **IO cost:** buffer gets = 4, logical reads = 20.

Složený klíč – plán bez použití indexu

- Lexikografickému uspořádání **neodpovídá** tento dotaz (chybí hodnoty atributu `idOrder`):

```
select * from OrderItems
where idProduct=47506; -- 13 resp. 9 zaznamu
```

- **QEP:** V B-stromu by bylo možné získat výsledek pouze sekvenčním průchodem, DBS raději zvolí sekvenční průchod haldou.
  - Oracle:

```
SELECT STATEMENT
  TABLE ACCESS FULL  ORDERITEMS
```

  - SQL Server:

```
|--Table Scan(OBJECT:(...[OrderItems]),...)
```

- **IO cost:** buffer gets = 3 184, logical reads = 3 760.

Složený klíč – specifikace dotazu

1. Lexikografickému uspořádání pro klíč $a_1, a_2, ..., a_k$ **odpovídají dotazy** obsahující bodové dotazy pro atributy $a_1, ..., a_l$, $l \leq k$, pro atribut a_{l+1} může být specifikován rozsah, atributy $a_{l+2}, ..., a_k$ mohou zůstat nespecifikované.
2. Jakýkoli jiný dotaz znamená nevyužití indexu, tedy vykonání dotazu sekvenčním průchodem haldou.
3. Pokud DBS odhadne velikost výsledku dle bodu 1 za příliš vysokou, provede dotaz dle bodu 2.

Složený klíč – plán využívající index

- Lexikografickému uspořádání složeného klíče (`idOrder`, `idProduct`) **odpovídá** tedy i tento dotaz:

```
select * from OrderItems
where idOrder=12456 and
idProduct between 20000 and 30000; -- 4 zaznamy
```

- **QEP:**
  - Oracle:

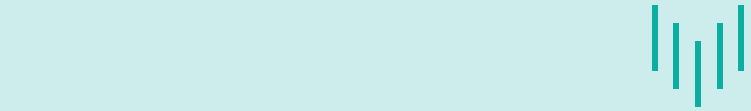
```
SELECT STATEMENT
  TABLE ACCESS BY INDEX ROWID BATCHED  ORDERITEMS
    INDEX RANGE SCAN                   SYS_C00470736
```

  - SQL Server:

```
|--Nested Loops(...)
   |--Index Seek(OBJECT:(...[PK__OrderIte__CD443163A0C75C37])
   |--RID Lookup(OBJECT:(...[OrderItems]), ...)
```

- **IO cost:** buffer gets = 4, logical reads = 7.

Výkon ekvivalentních dotazů, příklad



```
select f.film_id, f.title,
  count(distinct fa.actor_id) actor_count,
  count(distinct fc.category_id) category_count
from film f
left join film_actor fa on f.film_id=fa.film_id
left join film_category fc on f.film_id=fc.film_id
group by f.film_id, f.title
order by f.film_id
```

```
select f.film_id, f.title,
  (select count(*) from film_actor fa
   where f.film_id=fa.film_id) actor_count,
  (select count(*) from film_category fc
   where f.film_id=fc.film_id) category_count
from film f
order by f.film_id
```

U prvního řešení dostaneme CPU time = 109 ms, u druhého 0ms.

Reference

- Thomas Kyte. Expert Oracle Database Architecture: 9i and 10g Programming Techniques and Solutions.
- Oracle. CREATE TABLE manual.
  `https://docs.oracle.com/en/database/oracle/oracle-database/21/sqlrf/CREATE-TABLE.html`
- Oracle. CREATE INDEX manual.
  `https://docs.oracle.com/en/database/oracle/oracle-database/21/sqlrf/CREATE-INDEX.html`
- Microsoft. CREATE TABLE manual.
  `https://learn.microsoft.com/en-us/sql/t-sql/statements/create-table-transact-sql`
- Microsoft. CREATE INDEX manual.
  `https://learn.microsoft.com/en-us/sql/t-sql/statements/create-index-transact-sql`