Database and Information Systems

Michal Krátký

Department of Computer Science
Faculty of Electrical Engineering and Computer Science
VŠB – Technical University of Ostrava

2020/2021

Content

Physical implementation of DBMS

- Physical implementation of DBMS defines data structures for a storage of basic objects of DBMS:
  - Tables
  - Indices
  - Materialized view
  - Data partitioning
- Evidently, it is the lowest layer of a database system.
- There are many available data structures with some default options.
- For example: a heap table is often created after `CREATE TABLE` is sent, B^+-tree is often created after `CREATE INDEX` is sent.

Table

- *Heap table* – records are not sorted, it is often a default option of CREATE TABLE.
- *Data clustering* – records are sorted according a key.

Heap table 1/2

- The basic type of a table, it is often created after CREATE TABLE is sent.
- It is a **persistent paged array** stored in a file: it includes blocks of the size 8 kB, each block includes records.
- Records are not sorted:
  - Records are not directly **deleted**, they are only marked as deleted and we need a special statement to physically delete them (so called *shrinking*).
  - In the case of **insert**, a record is stored in the first empty position or in the end of the array.

Heap table 2/2

- Each record of the table has a unique number called *ROWID*, this number is not however stored in the table.
- It provides:
  - a sequence search in the case of SELECT ($O(n)$ complexity).
  - the best complexity of INSERT ($O(c)$).
  - In the case of DELETE and UPDATE, the record(s) has/have to be often found before the the operation is executed.

Data clustering

- Records are sorted according to a key – a variant of the B-tree is often used: INSERT, SELECT (point query), DELETE, UPDATE are in $O(\log n)$, SELECT (range query) is in $O(n)$.
- Leaf nodes of the tree include values of a key as well as other attributes of the table, in opposite to an index where only a key together with ROWID are stored.
- *Advantage:* More efficient searching of key values compared to the clustered table.
- *Disadvantage:* Records must be sorted.
- Oracle: **indexed organized table** (IOT), SQL Server: **clustered index** (default for CREATE TABLE).

Index Types

An index enables 'fast' searching of a key, ROWID then references the complete record in a heap table.

- *Simple index* – an index for one attribute.
- *Composite index* – its key includes more than one attribute.

An index is often implemented with the B^+*-tree* – it provides $O(\log n)$ for all operations (only the range query has $O(n)$).

Use Case

- ```
  CREATE TABLE User (
     login     VARCHAR(7) PRIMARY KEY,
     fname     VARCHAR(20) NOT NULL,
     lname     VARCHAR(20) NOT NULL,
     email     VARCHAR(30) NOT NULL);
  ```

  A heap table is created in the case of Oracle. A clustered table (the B-tree) is created for SQL Server.

- ```
  CREATE INDEX user_login ON User(login);
  ```

  - A B^+-tree is created, it includes `login` as a key and ROWID (a pointer to a record of the heap or clustered table).
  - This kind of index (the index for the primary key) is often created automatically after a heap table is created.

Oracle vs SQL Server

	Oracle	**SQL Server**
Table	Heap Table	Heap Table
Data clustering	Index organized table	Clustered index
B^+-tree	Index	Unclustered index

Motivation

Query:

```
SELECT Course.* FROM Student, Course, Student_Course
  WHERE Student.lname='Sobota' AND
        Student.login=Student_Course.login AND
        Student_Course.rok = 2009 AND
        Student_Course.idCourse = Course.idCourse
```

1. What is the order of the operations during the query processing?
2. Can we influence this order and can we influence the algorithms which are used during the query processing?

Query Plan Selection

- Appropriate query plan is selected by **a query optimizer**.
- The selection of the operations' order is a part of the query evaluation plan.
- Can we influence the query processing time?
  - There are several techniques for that: using parametrized queries, bulk operations, set-up of transactions.
  - On a database level we can select an appropriate physical design of our database.

Example

Suplier D

D#	Name	Country
D1	IBM	US
D2	Oracle	US
D3	Microsoft	US

Product P

P#	Type	Selling
P1	RDBMS	1 000
P2	OS	1 000
P3	IDE	100

SuplierProduct DP

D#	P#	Selling
D1	P1	200
D1	P2	100
D1	P3	10
D2	P1	350
D3	P1	200
D3	P2	900
D3	P3	80

Example

Query: "Return the names of those suppliers which supply the product P1"

((DP JOIN D) `WHERE` P# = P# (`'P1'`)) { Name }

((DP $\bowtie$ D) $\sigma_{P\#='P1'}$) Π_{Name}

DP $\bowtie$ D

D#	P#	Selling	Name	Country
D1	P1	200	IBM	US
D1	P2	100	IBM	US
D1	P3	10	IBM	US
D2	P1	350	Oracle	US
D3	P1	200	Microsoft	US
D3	P2	900	Microsoft	US
D3	P3	80	Microsoft	US

((DP $\bowtie$ D) $\sigma_{P\#='P1'}$) Π_{Name}

Name
IBM
Oracle
Microsoft

Example

- *Query:* "Return the names of those suppliers which supply the product P1"
- *Expression:* ((DP $\bowtie$ D) $\sigma_{P\#='P1'}$) Π_{Name}
- Database contains 100 suppliers, 10 000 records in table DP where only 500 relate to product P1.
- For simplicity we consider that the relation DP and D are represented by two files on disk.

Query Processing, version 1

Naive query processing (without optimization):

1. Join of DP and P: this step includes reading 100 suppliers 10 000×. The result containing 10 000 records is written to disk [1].
2. The selection `P#='P1'`: we read the 10 000 records from the previous step and we select 50 records.
3. Projection on `Name`: result contains 50 records.

[1] The problem of a large intermediate result is that it does not have to fit into the main memory.

Query Processing, version 2

The following steps lead to the same result, however, more effectively:

1. Selection: we select only rows in the DP table containing the P1 product (the result contains 50 records).
2. Join of the intermediate result and table D (the result contains 50 records).
3. Projection and the duplicities elimination (the result contains 50 records).

Query processing, Physical Implementation

- Let us consider one disk access for reading one data page (or block).
- Data page has 2 kB (DBMS often use 8kB).
- For simplicity consider that the page contains 100 records (in the case of both tables).
- Therefore, the DP table is stored in 100 pages and the table D is stored in one page.
- We speak about **I/O cost** or **logical accesses**.

Query processing, discussion 1/2

- In the first version, we process 302 of disk accesses and in the second version we process 104 of disk accesses.
- In the first version, we perform join 10^6 times whereas the second version performs only 5×10^3 joins (so called **CPU cost**)
- Even though both versions return the same result, the second version is more than $3\times$ efficient from the I/O cost point of view (and $1,000\times$ more efficient from the operation point of view).
- Next, we can create an index for the attribute `Name` ...

Comparison of plans

	((DP $\bowtie$ D) $\sigma_{P\#='P1'}$) Π_{Name}	((DP $\sigma_{P\#='P1'}$) $\bowtie$ D) Π_{Name}
I/O Cost	302	104
CPU Cost	1 030 000	10 100

Query processing, discussion 2/2

- We can see that some optimizations are done by the optimizer itself.
- However a user can create indices, table types (in general: the physical design of a database) and the optimizer then utilizes them.
- **Therefore, we should care about the physical database design if we want an efficient information system**.

Generation of Query Evaluation Plan

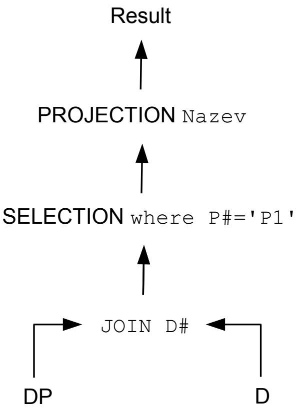
We identify 3 phases of query processing:

1. Transformation of a query into the internal form.
2. Transformation into a canonic form.
3. Generation of query plans and selection of the best query evaluation plan.

1. Transformation of a query into the internal form

- Transformation of an original query into some internal representation.
- We eliminate the syntax of the query language (SQL). Syntax and semantic control.
- We replace views by their definition.
- Internal form is usually some type of *query tree*.

Query tree



Since the query tree can be understood as a representation of an expression relational algebra, we will write it in a relational algebra in the following slides.

2. Transformation into canonic form 1/2

- Optimizer does a lot of transformations in this step.
- Relational algebra allows us to express the query many different ways.
- The transformation into the canonic form removes the insignificant differences in queries.

2. Transformation into canonic form 2/2

- Optimizer try to apply different transformation rules, which convert the expression into equivalent one.
  For example
  `(A JOIN B) WHERE selection on A`
  can be transformed into an equivalent one:
  `(A WHERE selection on A) JOIN B`
- **Query is not processed exactly how user write it!**
- We sometimes speak about a *query rewrite*.

3. Generation of query plans

- Optimizer creates a set of query plans.
- Optimizer calculates the cost of each generated query plan (usually the cost sum).
- The algorithm cost is dependent on the table size, intermediate result size and many other statistics (build by a DBMS).
- (Estimation of the intermediate result size is often problematic.)
- The optimizer selects the best (cheapest) query evaluation plan and it is processed.

Query Processing Plan

In a DBMS we can see the query processing plan:

- Oracle:
  - It is stored in table `PLAN_TABLE` using the `explain plan` command, e.g.:
    ```
    explain plan for select * from student where
    surname='Poe';
    ```
  - SQL Developer can report the plan.
- SQL Server:
  - Management Studio can report the plan (using Show Execution Plan in a menu).

Table Creation

- Let us have a table student `Student`:

```
CREATE TABLE Student (
  login CHAR(6) PRIMARY KEY,
  fname VARCHAR2(30) NOT NULL,
  lname VARCHAR2(50) NOT NULL,
  email VARCHAR2(40),
  account NUMBER);
```

Query Processing Plan

Output contains each operation performed during the query processing (IO cost and CPU cost).

```
explain plan for select * from student where lname='Poe';
```

Output:

```
Operation                      Object
------------------------------ ------------------------------
SELECT STATEMENT ()
 TABLE ACCESS (FULL)           STUDENT
```

It means that a sequential scan in a heap table is processed.

Query Plan Operations[2]

- Logical operations:
  - Selection
  - Projection
  - Join
  - Sort
- Physical operations:
  - TABLE ACCESS (FULL)
  - INDEX (UNIQUE SCAN ane RANGE SCAN)

[2] These operations are for Oracle, however SQL Server provides similar operations.

Example

```
CREATE TABLE Producer (
  id INT PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  address VARCHAR(50) NOT NULL)

CREATE TABLE Store_item (
  id INT PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  idProducer INT REFERENCES Producer NOT NULL)
```

Example

- Table `Producer` contains 100 000 records: 512 blocks[3][4], average record length is 24B, average number of records on a block is 195.
- Table `Store_item` contains 1 000 000 records: 4 352 blocks, avg. length of a record is 21B, average number of records on a block is 230.

[3] select blocks from user_segments where segment_name = 'PRODUCER';
[4] The block size is 8kB.

TABLE ACCESS (FULL)

Meaning: Sequential scan of a table - all blocks are accessed.

When it occurs? `SELECT * FROM Store_item;`

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			1178
TABLE ACCESS	STORE_ITEM	FULL	1178

TABLE ACCESS (FULL) with a selection

Meaning: Sequential scan of a table and a condition processing.

When it occurs?

```
SELECT * FROM Store_item WHERE name='PRA-2010-100000';
```

if the attribute `name` is not indexed.

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			1180
TABLE ACCESS	STORE_ITEM	FULL	1180
Filter Predicates			
NAME='PRA-2010-10000'			

Consequence: Low query efficiency.

INDEX (UNIQUE SCAN)

Meaning: Searching for one key in the index.

When it occurs?

```
SELECT id FROM Store_item WHERE id=50000;
```

if the attribute `id` is indexed (primary keys are indexed automatically).

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			1
INDEX	SYS_C00202648	UNIQUE SCAN	1
Access Predicates			
ID=50000			

INDEX (RANGE SCAN)

Meaning: Searching for a range of keys in the index.

When it occurs?

```
SELECT id FROM Store_item WHERE id > 1 AND id < 10000;
```

if the attribute `id` is indexed.

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			25
INDEX	SYS_C00202648	RANGE SCAN	25
Access Predicates			
AND			
ID>1			
ID<10000			

Notice: RANGE SCAN is processed even if the attribute is not unique.

TABLE ACCESS (BY INDEX ROWID)

Meaning: Access to one record in table, which follow after the index search.

When it occurs?

```
SELECT name FROM Store_item WHERE id=50000;
```

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			1
TABLE ACCESS	STORE_ITEM	BY INDEX ROWID	1
INDEX	SYS_C00202648	UNIQUE SCAN	1
Access Predicates			
ID=50000			

JOIN 1/2

Meaning: Join of two tables.

When it occurs?

```
SELECT S.id, S.name, P.name FROM Store_item S,Producer P WHERE
S.name='PRA-2010-10000' and S.idProducer=P.id;
```

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			1220
NESTED LOOPS			
NESTED LOOPS			1220
TABLE ACCESS	STORE_ITEM	FULL	1177
Filter Predicates			
STORE_ITEM.NAME='PRA-2010-10000'			
INDEX	SYS_C00203250	UNIQUE SCAN	0
Access Predicates			
STORE_ITEM.IDPRODUCER=PRODUCER.			
TABLE ACCESS	PRODUCER	BY INDEX ROWID	1

Notice: Join is performed by the nested loop algorithm.

JOIN 2/2

Meaning: Join of two tables without access into the index.

When it occurs?

```
SELECT S.id, S.name, P.name FROM Store_item S,Producer P WHERE
S.name='PRA-2010-10000' and S.idProducer=P.id;
```

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			1155
HASH JOIN			1155
Access Predicates			
STORE_ITEM.IDPRODUCER=PRODUCER.ID			
TABLE ACCESS	STORE_ITEM	FULL	1141
Filter Predicates			
STORE_ITEM.NAME='PRA-2010-10000'			
TABLE ACCESS	PRODUCER	FULL	13

Notice: Join is performed by the Hash join algorithm.

Sorting

Query: `SELECT * from Store_item ORDER BY name;`

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			11422
SORT		ORDER BY	11422
TABLE ACCESS	STORE_ITEM	FULL	1139

Notice: Index could avoid sortin only in the case of SQL command `SELECT name from Store_item ORDER BY name`.

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			7033
SORT		ORDER BY	7033
INDEX	STORE_ITEM_NAME	FAST FULL SCAN	1048

Simple Index

- We create an index on the attribute `name` of the table `Producer` (it contains 100 000 records):
  `CREATE INDEX Producer_name ON Producer(name);`
- The index size is 60% of the table size (384 blocks[5], the table size is 640 blocks).
- *Query:* `SELECT * FROM Producer WHERE name='prod7452';`
  The result size: 1
- Logical accesses: 3

[5] `SELECT blocks FROM user_segments WHERE segment_name = 'PRODUCER_NAME';`

Composite Index

- When we want to query two and more attributes in one query, we can create a *composite index* (or an *index with compound key*).
- For example: `CREATE INDEX Producer_name_addr ON Producer(name,address);`
- The index size is 80% of the table size (512 blocks[6] , the size of the index Producer(name) is 384 blocks, the table size is 640 blocks).
- *Query:* `SELECT * FROM Producer WHERE name='prod7452' AND address='address56000';`
- Logical access: 3

[6] `SELECT blocks FROM user_segments WHERE segment_name = 'PRODUCER_NAME_ADDR';`

Querying of Individual Attributes 1/3

- *Query:* `SELECT * FROM Producer WHERE name='prod7452';`

OPERATION	OBJECT_NAME	COST
SELECT STATEMENT		3
TABLE ACCESS (BY INDEX ROWID)	PRODUCER	3
INDEX (RANGE SCAN)	PRODUCER_NAME_ADDR	2
Access Predicates		
NAME='prod7452'		

Logical access: 3

- *Query:* `SELECT * FROM Producer WHERE address='address56000';`

OPERATION	OBJECT_NAME	COST	LAST_CR_BUFFER_GETS
SELECT STATEMENT		172	
TABLE ACCESS (FULL)	PRODUCER	172	552
Filter Predicates			
ADDRESS='address56000'			

Logical access: 574 !!!

Querying of Individual Attributes 2/3

- Although the attribute `address` is a part of the compound key (`name,address`), the query is processed with a sequential scan in a heap table $\Rightarrow$ we get 574 instead of 3 logical access.
- **Why?** The composite index is implemented by the B-tree with the compound key (`name,address`) in that order. As a result it supports only queries for attributes `name` or (`name,address`) (these queries are processed using point or range queries in the B-tree).

Querying of Individual Attributes 3/3

Solution? When we query the attributes `name`, `(name,address)`, and `address`), we can create the composite index `(name,address)` and a simple index `(address)`.

Properties:

- The size of both indices is 140% of the table size (896 blocks, the table size is 640 blocks).
- The update of the attribute `name` (the operations INSERT, UPDATE, DELETE) means the update of the table and one index.
- However, the update of the attribute `address` means the update of one table and two indices.

Candidates for Index

- Index is often created for keys and foreign keys.
- Two main rules when an index should be created for a table:
  - The index is used to retrieve a low number of records of a table.
  - The index covers one or more queries.
- Each index means a high overhead of update operations!

Reference

- Thomas Kyte. Expert Oracle Database Architecture: 9i and 10g Programming Techniques and Solutions.
- Oracle. CREATE TABLE manual. http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/statements_7002.htm
- Oracle. CREATE CLUSTER manual. http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/statements_5001.htm
- Microsoft. Clustered Index Design Guidelines. http://msdn.microsoft.com/en-us/library/ms190639.aspx