Database management, advanced Indexes

Gergely Lukács

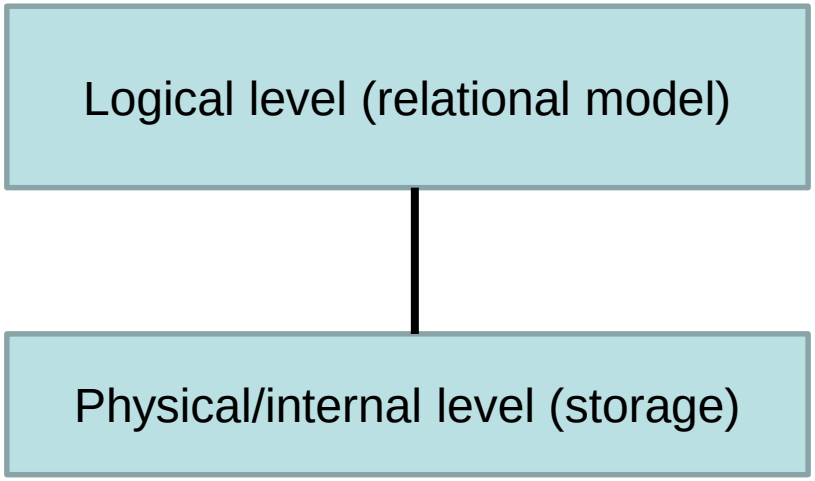
Pázmány Péter Catholic University

Faculty of Information Technology and Bionics

Budapest, Hungary

lukacs@itk.ppke.hu

DB architecture (fraction)

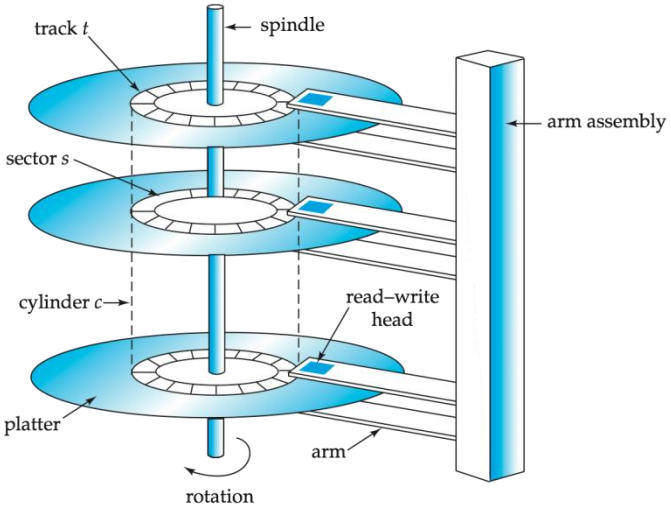


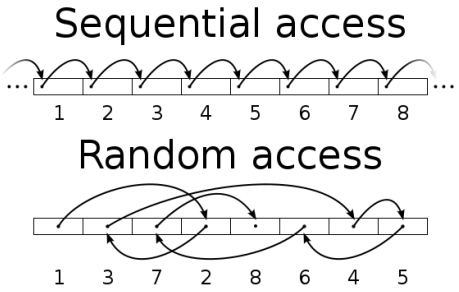
Physical data independence

Data access structures,

„Index“ (B+ tree)

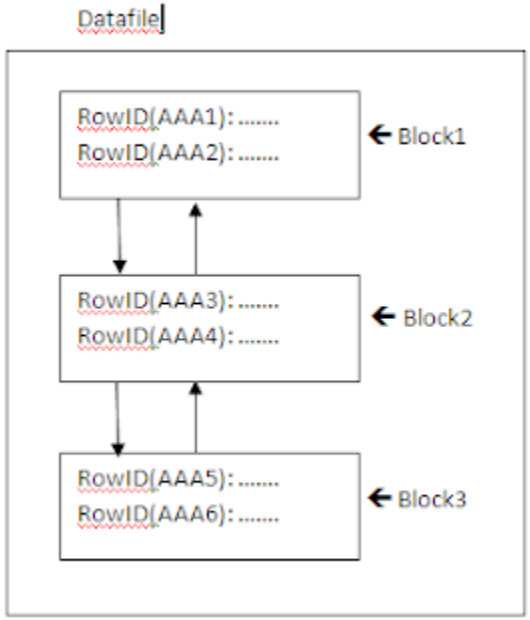
Disk, Data organisation, data access costs





- Sequential access much faster, than random access
- **Block** – a contiguous sequence of sectors from a single track
  - data is transferred between disk and main memory in blocks
  - Sizes e.g, 4/8/16/32 kB,
    - Smaller blocks: more transfers from disk
    - Larger blocks: more space wasted due to partially filled blocks
    - Typical block sizes today range from 4 to 16 kilobytes
- Significant cost factor in DBMS: # blocks accessed from disk

RDBMS: storing tables: blocks on a disk



„Row store”,
record by record

Index Motivation

- What about if we want to be able to search quickly along an attribute?
- We'll create separate data structures called ***indexes*** to address all these points

Index
(https://en.wikipedia.org/wiki/Index)

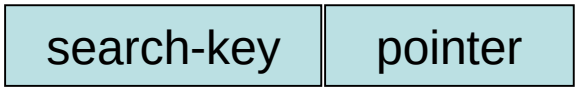
Science, technology, and mathematics [edit]

Computer science [edit]

- Index, a key in an associative array
- Index, a character in Unicode, its code is 132
- Index, the dataset maintained by search engine indexing
- Array index, an integer pointer into an array data structure
- BitTorrent index, a list of torrent files available for searches
- Database index, a data structure that improves the speed of data retrieval
- Index mapping of raw data for an array
- Index register, a processor register used for modifying operand addresses during the run of a program
- Indexed color, in computer imagery
- Indexed Sequential Access Method (ISAM), used for indexing data for fast retrieval
- Lookup table, a data structure used to store precomputed information
- Site map, or site index, a list of pages of a web site accessible to crawlers or users
- Subject indexing, describing the content of a document by keywords
- Web indexing, Internet indexing
- Webserver directory index, a default or index web page in a directory on a web server, such as index.html

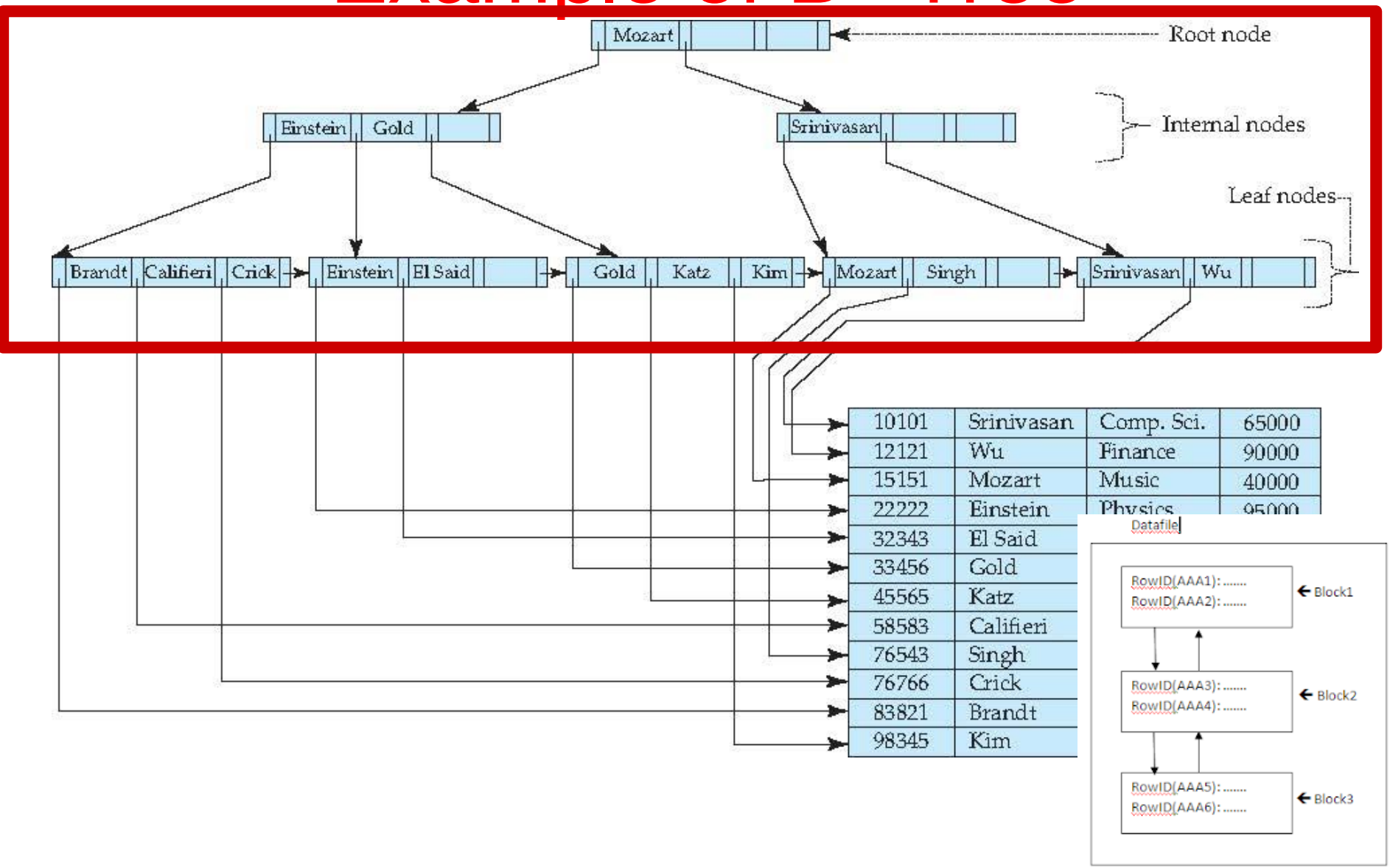
Basic Concepts

- Indexing mechanisms used to speed up access to desired data.
- **Search Key** - attribute (or set of attributes) used to look up records in a file.
- An **index (file)** consists of **index entries** of the form

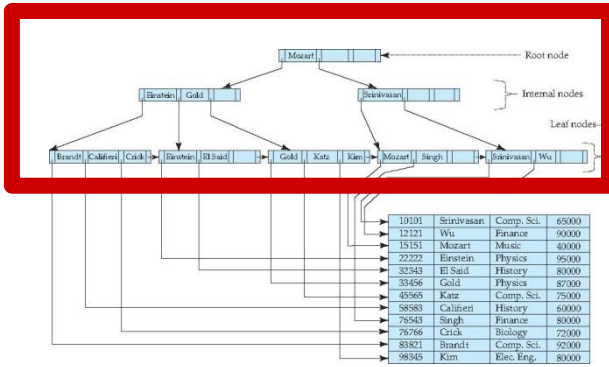


- Index files support finding an index entry for a search key efficiently (and are typically smaller than the original file)
- Two basic kinds of indices:
  - **Ordered indices:** search keys are stored in sorted order in some kind of a search tree
  - **Hash indices:** search keys are distributed uniformly across "buckets" using a "hash function".

Example of B+-Tree

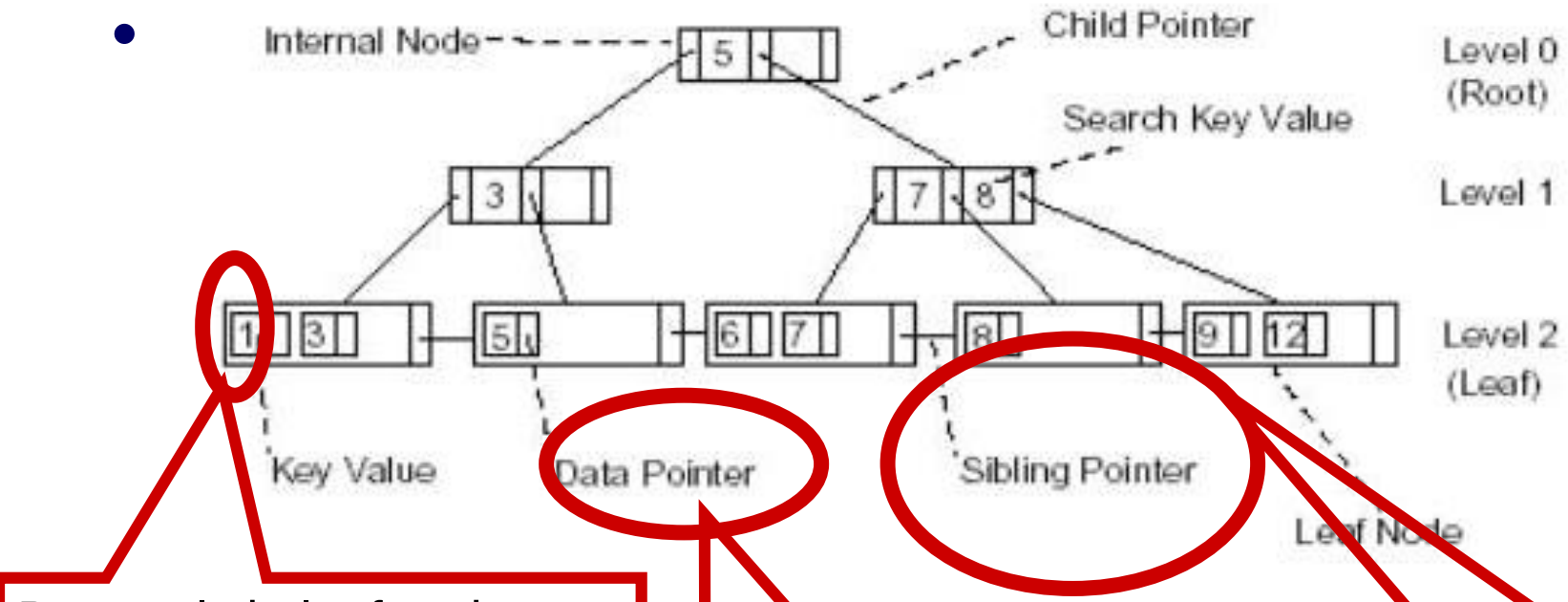


B+ tree



- B (balanced) tree
-
-

Size of node ~ block on disk



Data only in leaf nodes

Pointer on the data (table)

Leaf nodes chained (for fast sequential traversar)

Typical size of index

- Node ~ block on disk
  (block ~ 4kb,…, 32kb)
- Number of children of a node? (order of magnitude?)
- Depth of an index?
  - How does it change as the number of records increases? (function?)

Oracle: Data pointer: ROWID

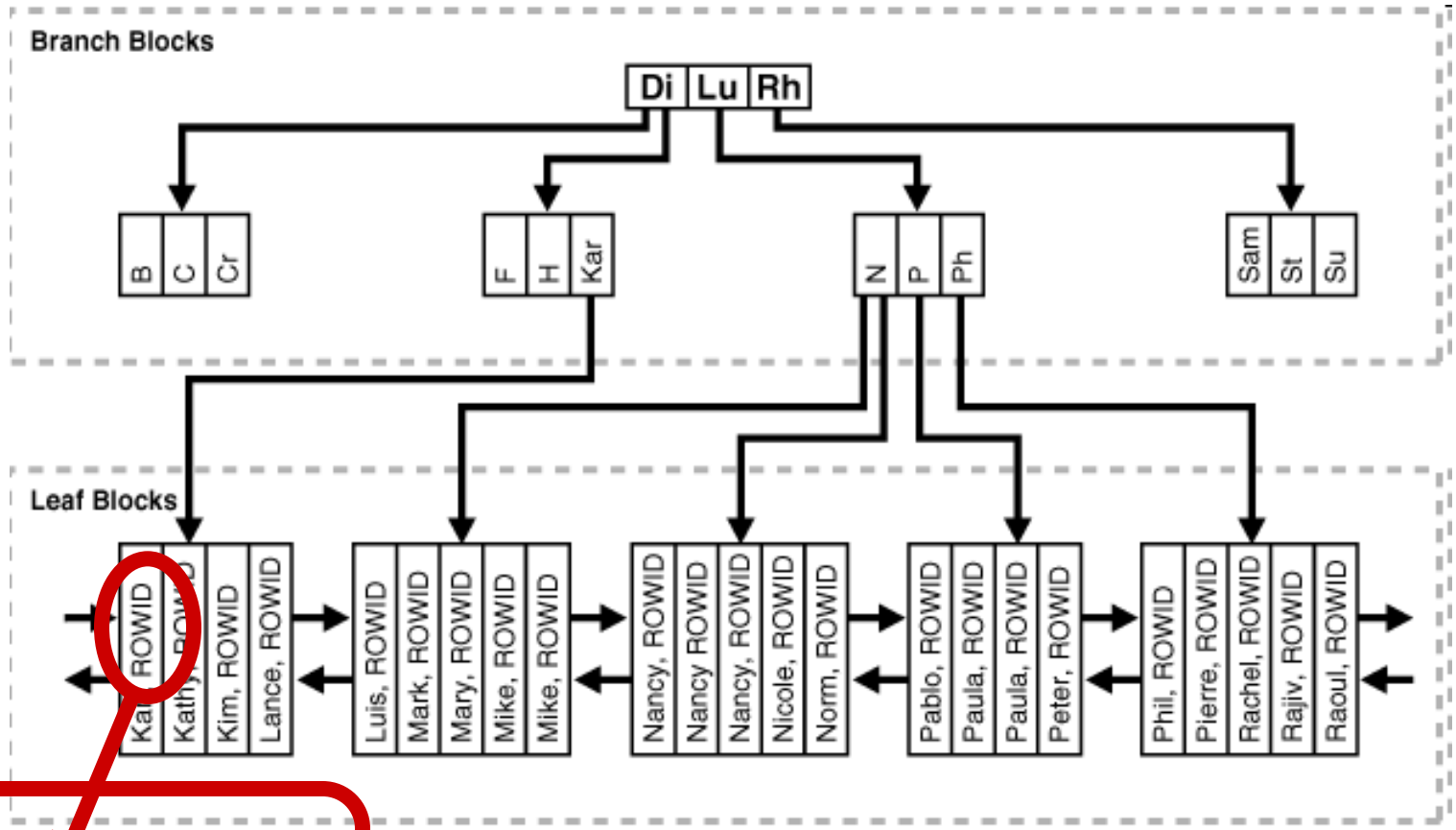


Table (record)

SQL: CREATE INDEX

```
CREATE INDEX emp_ename
ON emp(ename);

CREATE INDEX ci ON c (c1, c2);
```

Index useful for

- Query conditions
  - Exact match query, range query
  - Selectivity ! (rule of thumb: 5% selectivity!)
    - see later, clustering factor
- Sorted result
- Join conditions (next lecture)

B^+-Tree Index Files

- Advantages of B^+-tree index files:
  - automatically reorganizes itself with small, local, changes, in the face of insertions and deletions.
  - Reorganization of entire file is not required to maintain performance.
  - ((compacting,rebuilding … can make sense))
- Disadvantages of B^+-trees:
  - extra insertion and deletion overhead
    - Finding records to update or delete can be (significantly) faster with an index
  - space overhead (size of index comparable to size of table)
- B^+-trees are used extensively

Indexes on Multiple Attributes

- **Composite search keys** are search keys containing more than one attribute
  - E.g. (*dept_name, salary*)
- Lexicographic ordering: $(a_1, a_2) < (b_1, b_2)$ if either
  - $a_1 < b_1$, or
  - $a_1 = b_1$ and $a_2 < b_2$

Indexes on Multiple Attributes 2

- **CREATE INDEX Idx_Emp_Name ON Employees ("Last Name", "First Name")**
- Useful for
  - ... WHERE "Last Name" = 'Doe'
  - ... WHERE "Last Name" = 'Doe' AND "First Name" = 'John'
  - ... WHERE "First Name" = 'John' AND "Last Name" = 'Doe'
- Can not be used
  - ... WHERE "First Name" = 'John'

Indexes on Multiple Attributes 3

Suppose we have an index on combined search-key (*dept_name, salary*).

- **where** *dept_name* = “Finance” **and** *salary* = 80000
  the index can be used to fetch only records that satisfy both conditions.
- Can also efficiently handle
  where *dept_name* = “Finance” **and** *salary* < 80000
- But cannot efficiently handle
  where *dept_name* < “Finance” **and** *balance* = 80000

Covering index (for a specific query)

- Add extra attributes to index so (some) queries can avoid fetching the actual records
- We say that an index is *covering* for a specific query if the index contains all the needed attributes- meaning the query can be answered using the index alone!

Multiple conditions, multiple indices

- Use multiple indices for certain types of queries.
- Example:

```
SELECT id
  FROM instructor
 WHERE dept_name = ”finance”
   AND salary = 80000;
```

Possible strategies for processing query using indices on single attributes:

- Using index on dept_name
  1. Use index on *dept_name* to find instructors with department name Finance;
  2. test *salary* = *80000*
- Use index on *salary*
  1. to find instructors with a salary of $80000;
  2. test *dept_name* = “Finance”.
- Use both indices (Oracle?!)
  1. *dept_name* index to find pointers to all records pertaining to the “Finance” department.
  2. Similarly use index on *salary*.
  3. Take intersection of both sets of pointers obtained.

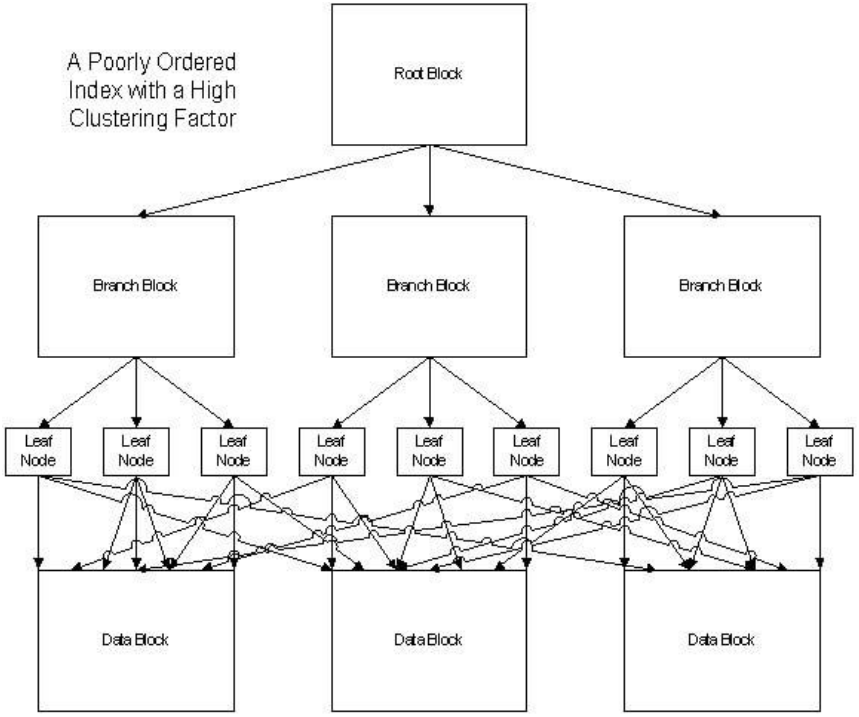
Bulk Loading and Bottom-Up Build

- Inserting entries one-at-a-time into a B^+-tree requires $\geq$ 1 IO per entry
  - assuming leaf level does not fit in memory
  - can be very inefficient for loading a large number of entries at a time (**bulk loading**)
- Efficient alternative 1: **Bottom-up B^+-tree construction**
  - Deactivating, dropping index
  - Reactivating, recreating
- Efficient alternative 2:
  - sort entries first
  - insert in sorted order
    - insertion will go to existing page (or cause a split)
    - much improved IO performance, but most leaf nodes half full

(Clustering factor)

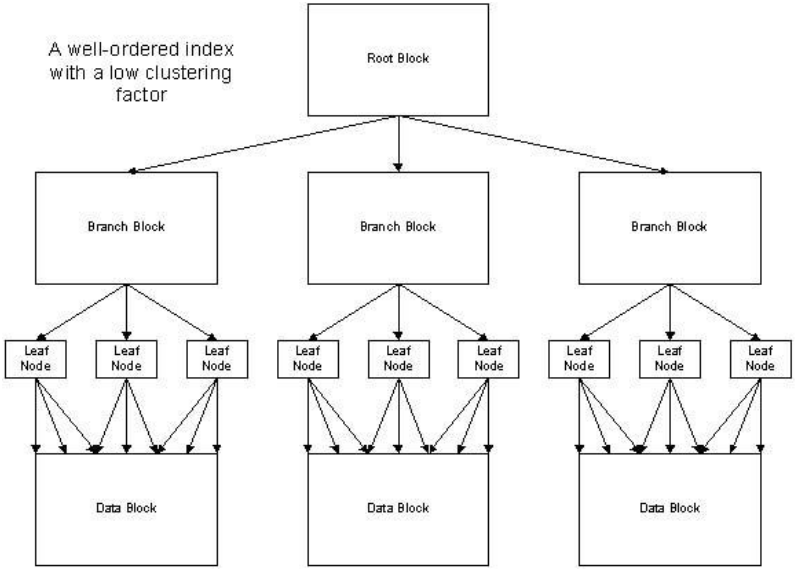
Poorly ordered table related to an index

- Oracle: „high clustering factor”;
- PostgreSQL: lowcorrelation

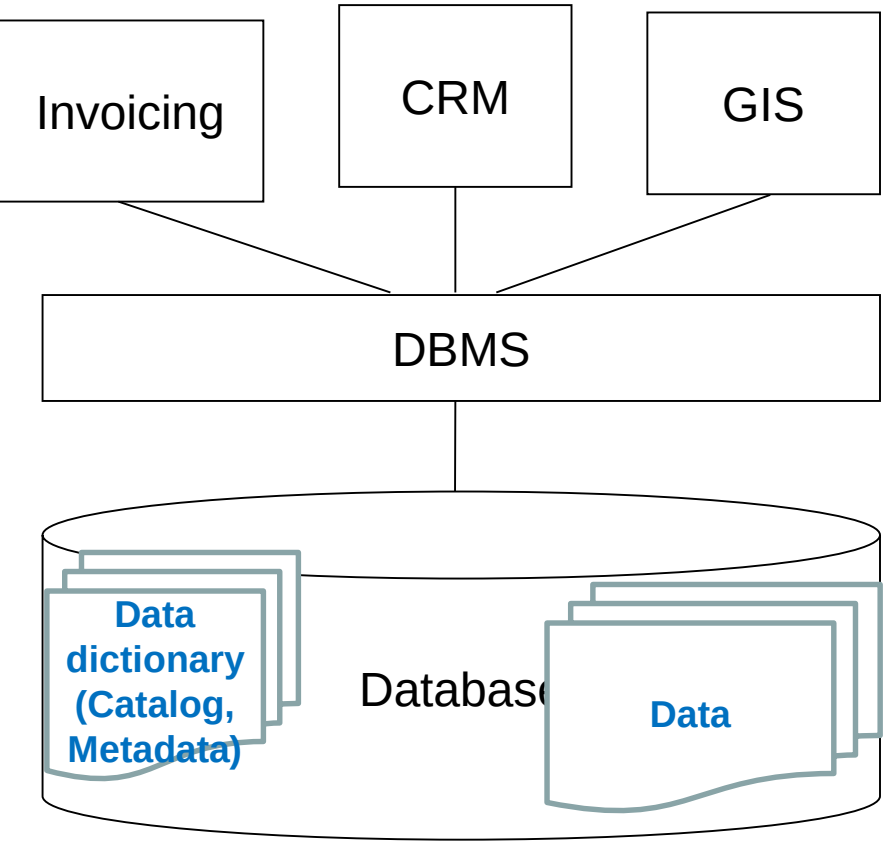


index is useful for a query condition with at most ~ ~ 5% selectivity! (rule of thumb)

Well ordered table related to an index



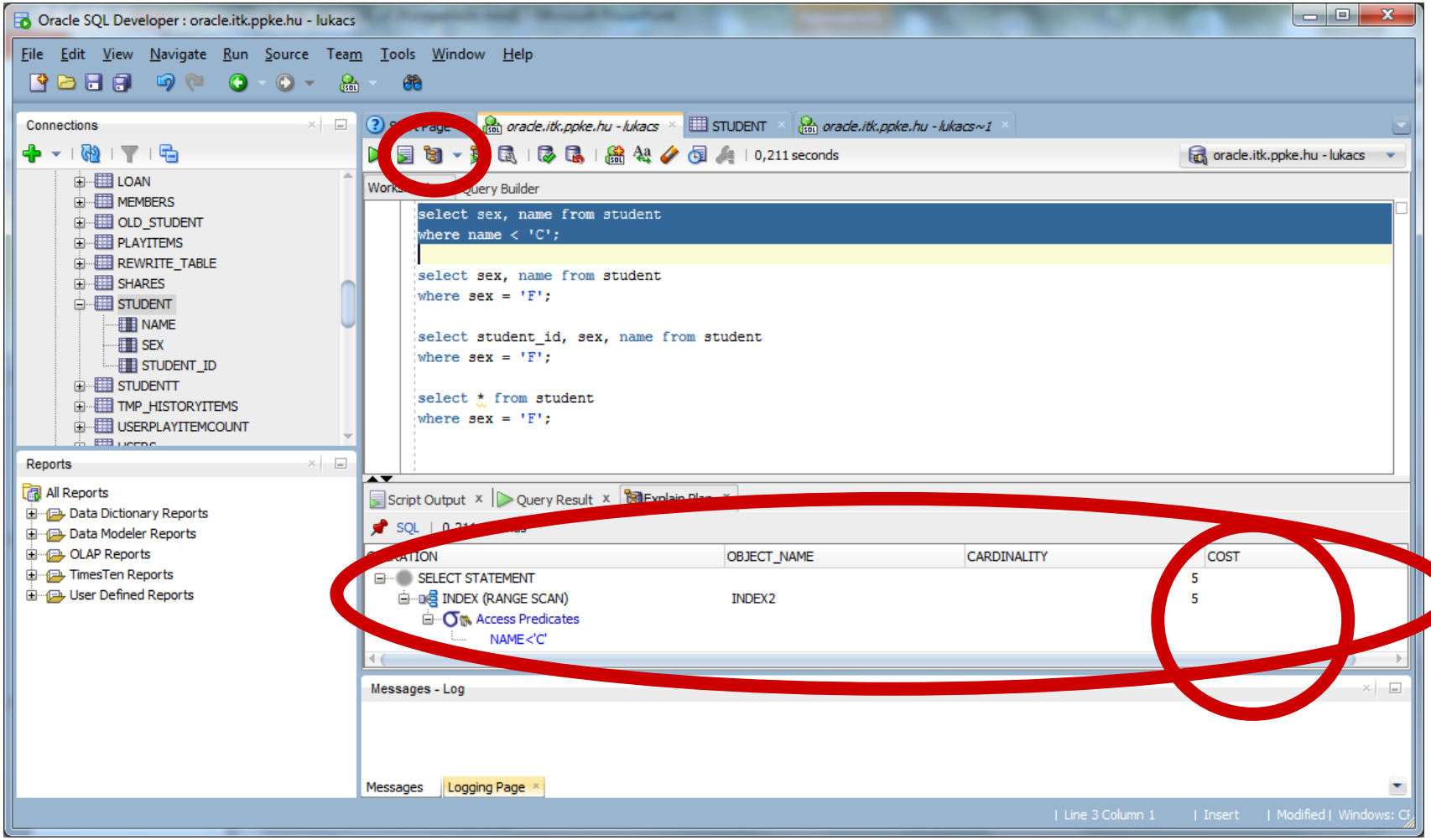
Data dictionary



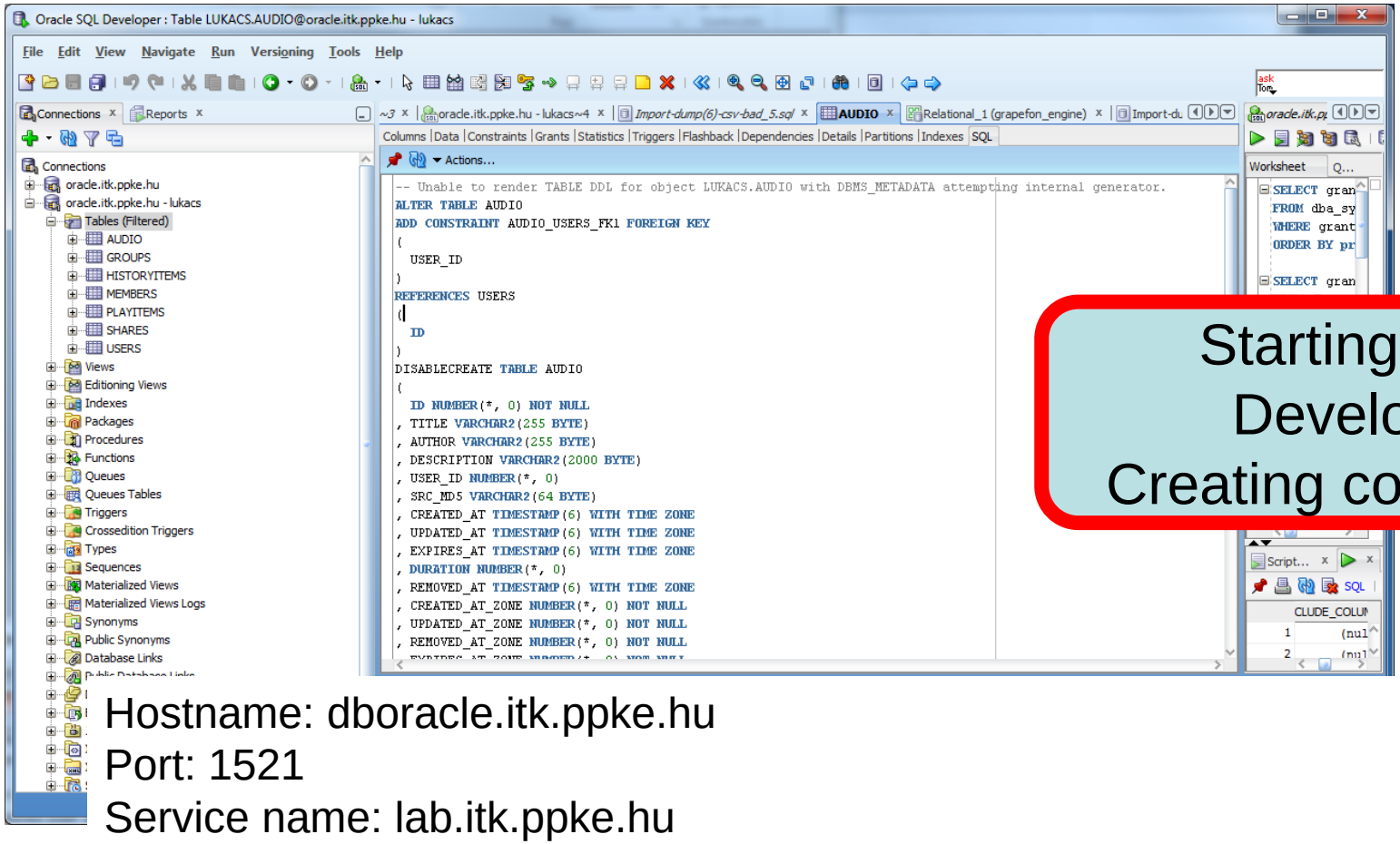
- Home / Database / Oracle Database Online Documentation 12c Release 1 (12.1) / Database Administration
- Database Reference
  - **all_tables**
  - **all_indexes (https://docs.oracle.com/database/121/REFRN/GUID-E39825BA-70AC-45D8-AF30-C7FF561373B6.htm#REFRN20088)**
  - **SQL> select value from v$parameter where name = 'db_block_size' ;**
    VALUE

    8192

Execution plan



SQL Developer



Starting SQL Developer
Creating connection

Hostname: dboracle.itk.ppke.hu
Port: 1521
Service name: lab.itk.ppke.hu
Username: AD19_FAMFIR (3-3 characters of family name and christian name)
Jelszó: ******

Creating and Dropping an Index

1. Create the student table using the appropriate script!
2. Query the ROWID!
3. Query the names starting with letter “B”!
    1. Result: only name attribute
    2. Result: all attributes („*”)
4. Check the execution plan in both cases!
5. Create an index on the name attribute (recommended index name: student_name_ix)!
6. Check the execution plan again!
7. Drop the index!

Covering Index for a Query

1. Query the attributes NAME and SEX!
2. Check the execution plan!
3. Create a covering index for the query and check the execution plan again!
4. Query the NAME and SEX attributes with the query criterion SEX=”F”!
5. Check the execution plan! How is the index used?
6. Query all attributes of the table with the query criterion SEX=”F”!
7. Check the execution plan!

Size of Index

Background information (only a db administrator can execute this query)

```
SELECT value
FROM   v$parameter
WHERE  name = 'db_block_size';
```

workaround for normal users:

```
SELECT DISTINCT bytes / blocks
FROM   user_segments;
```

https://stackoverflow.com/questions/4862724/query-my-block-size-oracle

1. Check the number of blocks in the `HISTORYITEMS_LARGE` table! How much space is needed for the table? (Is it OK to count only leaf blocks?!)
2. Check the indexes on the table, including their sizes!

HISTORYITEMS_LARGE – Index Usage Examples

1. Query records from the `AD18___DB.HISTORYITEMS_LARGE` for the first the 10 users without using an index!
   (Query hint: `SELECT /*+ NO_INDEX(HISTORYITEMS_LARGE) */ * FROM …`)
   Check the execution plan (cost of query!)!
2. Execute the same query with index usage! How did the execution cost change?
3. Query the records with a starttime between Jan. 2014 and March 2015! Check the execution plan!
4. Extend the previous query with an additional condition, restricting the answer to the first 100 users!
5. When, how and why are indexes used in the different situations?

Index Usage - Query Selectivity

Check the following query with different constant values in the `WHERE` condition (i.e., with different selectivities)!

```
SELECT Count(updated_at)
FROM   historyitems_large

WHERE  user_id < 50;
```

1. When (at which selectivities) does the DBMS use the index? How does the execution cost changes? (Create a notice in form of a table: paramater value, query selectivity, index usage (yes/no), execution cost.)
2. What changes, when querying COUNT(duration)? When querying SUM(duration)? Why?
   (In both cases, please use the `/*+ NO_REWRITE */` optimizer hint!)

Index Usage – Multiple Indixes

Check the execution plan of the following query!

```
SELECT Count(updated_at)
FROM   ad18___db.historyitems_large
WHERE  audio_id < 10
       AND Sys_extract_utc(started_at)
           < To_date('2011-07-25', 'yyyy-mm-dd');
```

1. Which index or indexes are used? Can you see a special operation?

Index Usage - Clustering Factor

1. The table HISTORYITEMS_LARGE_CF has exactly the same records as the table HISTORYITEMS_LARGE. Please check it!
2. Check the previous queries with the HISTORYITEMS_LARGE_CF table!
3. Are there any differences in the index usage you can observe?
4. Check the clustering factor of the indexes!

Index Usage – Query Covering All Rows

- Query the number of records per user!
- Does the DBMS use the index for executing the query?
  How (operation)?
  Explanation?
  Does the index support the query properly?

USER_ID	COUNT(*)
1	143
2	297
3	284
4	287
5	307
6	301

Statistics

Check the statistics kept on the data by Oracle!
ALL_TABLES and DBA_OBJECT_TABLES
ALL _TAB_STATISTICS and ALL _TAB_COL_STATISTICS
ALL _TAB_HISTOGRAMS
ALL _TAB_COLS
ALL _COL_GROUP_COLUMNS
ALL _INDEXES and ALL _IND_STATISTICS

Database management II.

Join Execution
Database optimisation

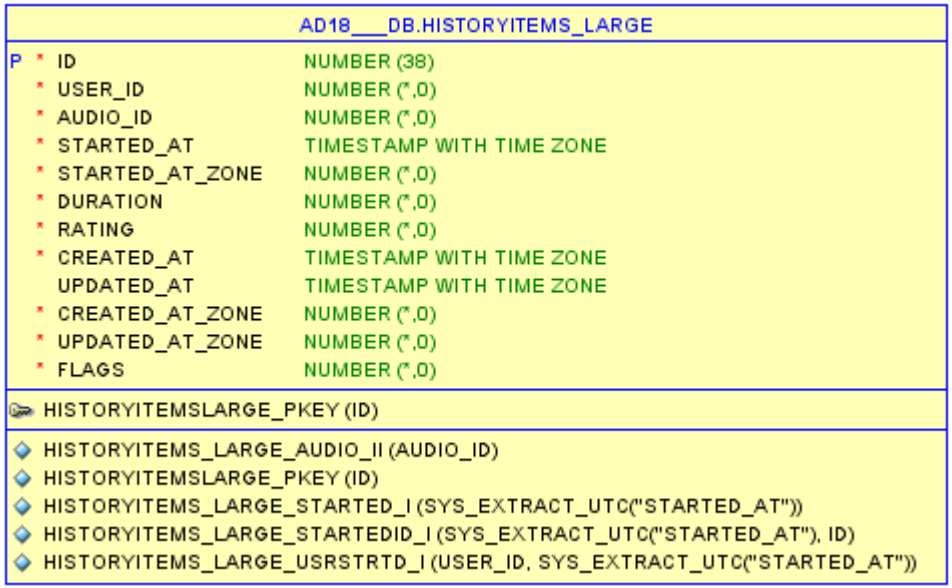
Gergely Lukács
Pázmány Péter Catholic University
Faculty of Information Technology and Bionics
Budapest, Hungary
lukacs@itk.ppke.hu

Index Usage - Query Selectivity

Check the following query with different constant values in the `WHERE` condition (i.e., with different selectivities)!

```
SELECT Count(updated_at)
FROM   ad18___db.historyitems_large

WHERE  user_id < 50;
```

1. When (at which selectivities) does the DBMS use the index? How does the execution cost changes? (Create a notice in form of a table: paramater value, query selectivity, index usage (yes/no), execution cost.)
2. What changes, when querying COUNT(duration)? When querying SUM(duration)? Why?
   (In both cases, please use the `/*+ NO_REWRITE */` optimizer hint!)


AD18___DB.HISTORYITEMS_LARGE
P * ID NUMBER (38)
* USER_ID NUMBER (*,0)
* AUDIO_ID NUMBER (*,0)
* STARTED_AT TIMESTAMP WITH TIME ZONE
* STARTED_AT_ZONE NUMBER (*,0)
* DURATION NUMBER (*,0)
* RATING NUMBER (*,0)
* CREATED_AT TIMESTAMP WITH TIME ZONE
UPDATED_AT TIMESTAMP WITH TIME ZONE
* CREATED_AT_ZONE NUMBER (*,0)
* UPDATED_AT_ZONE NUMBER (*,0)
* FLAGS NUMBER (*,0)
HISTORYITEMSLARGE_PKEY (ID)
HISTORYITEMS_LARGE_AUDIO_II (AUDIO_ID)
HISTORYITEMSLARGE_PKEY (ID)
HISTORYITEMS_LARGE_STARTED_I (SYS_EXTRACT_UTC("STARTED_AT"))
HISTORYITEMS_LARGE_STARTEDID_I (SYS_EXTRACT_UTC("STARTED_AT"), ID)
HISTORYITEMS_LARGE_USRSTRTD_I (USER_ID, SYS_EXTRACT_UTC("STARTED_AT"))

Data Access Structures, Indexes

```
select user_id, count(*) from lukacs.historyitems_large
 group by user_id;
```

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		9983	2382
HASH (GROUP BY)		9983	2382
INDEX (FAST FULL SCAN)	HISTORYITEMS_LARGE_USRSTRTD_I	2971500	1886

```
select user_id, count(*) from lukacs.historyitems_large
 group by user_id order by user_id;
```

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		9983	2382
SORT (GROUP BY)		9983	2382
INDEX (FAST FULL SCAN)	HISTORYITEMS_LARGE_USRSTRTD_I	2971500	1886

Data warehouse queries

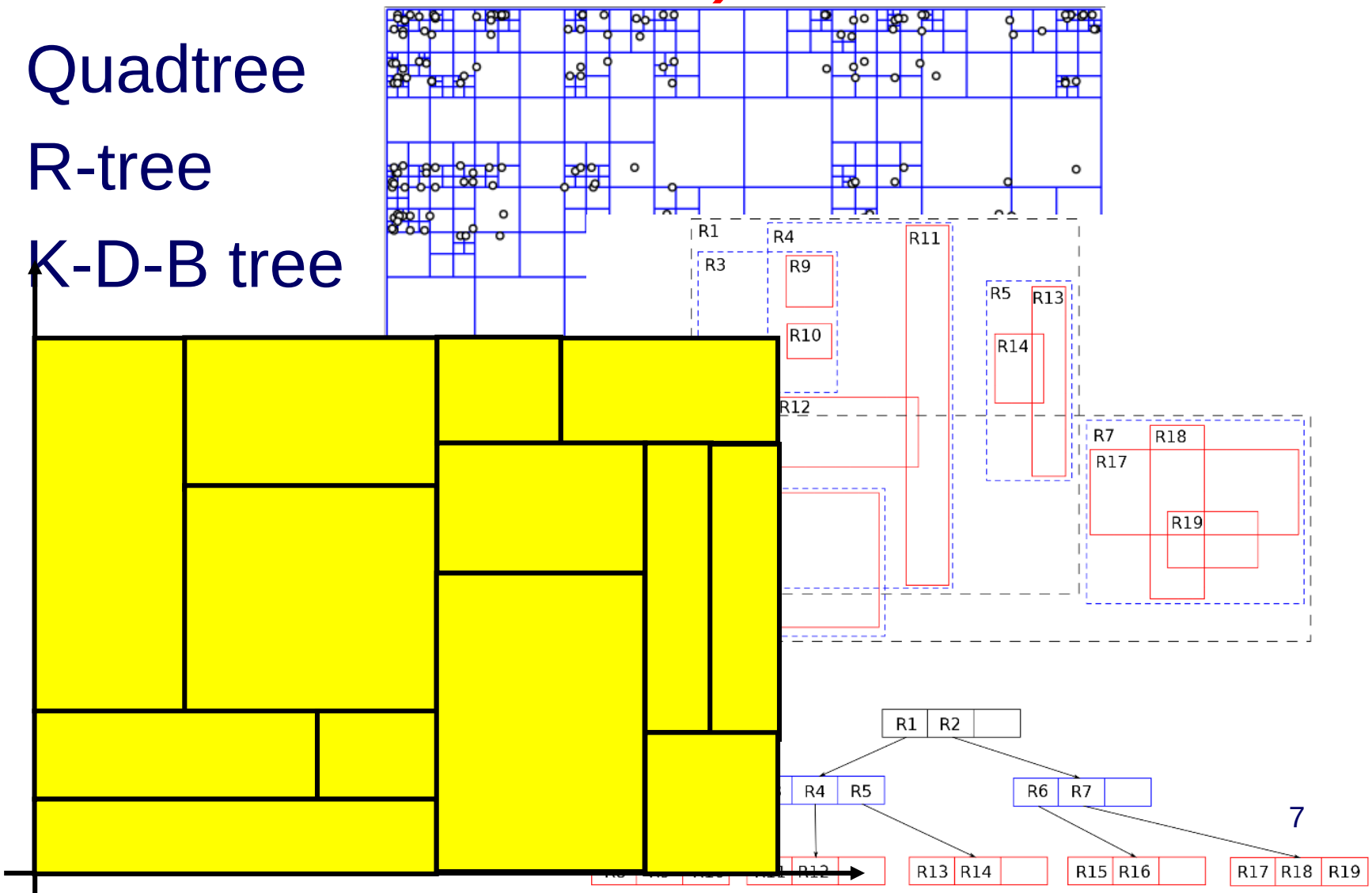
- Large number of records
- Ad-hoc queries, multiple dimensions
- Aggregations

- Read operations almost exclusively

- Bitmap index
- Materialized view + query rewrite

Geographic databases (> 1D data)

- Quadtree
- R-tree
- K-D-B tree



Join execution

8

Join execution

- Relational databases, normalisation
  => large number of joins
- Join – very expensive
- Several ways to execute

Nested loop join

Notes

- We are again considering “IO aware” algorithms: ***care about disk IO***

- Given a relation R, let:
  - T(R) = # of tuples in R
  - P(R) = # of pages in R

Recall that we read / write entire pages with disk IO

Nested Loop Join (NLJ)

Compute $R \bowtie S$ *on* A:

```
for r in R:
  for s in S:
    if r[A] == s[A]:
      yield (r,s)
```

Nested Loop Join (NLJ)

Compute $R \bowtie S$ *on* A:

```
for r in R:
  for s in S:
    if r[A] == s[A]:
      yield (r,s)
```

Cost:

P(R)

1. **Loop over the tuples in R**

Note that our IO cost is based on the number of ***pages*** loaded, not the number of tuples!

Nested Loop Join (NLJ)

```
Compute R ⋈ S on A:
  for r in R:
    for s in S:
      if r[A] == s[A]:
        yield (r,s)
```

Cost:

P(R) + T(R)*P(S)

1. Loop over the tuples in R
2. **For every tuple in R, loop over all the tuples in S**

Have to read ***all of S*** from disk for ***every tuple in R!***

Nested Loop Join (NLJ)

```
Compute R ⋈ S on A:
  for r in R:
    for s in S:
      if r[A] == s[A]:
        yield (r,s)
```

Cost:

P(R) + T(R)*P(S)

1. Loop over the tuples in R
2. For every tuple in R, loop over all the tuples in S
3. **Check against join conditions**

Note that NLJ can handle things other than equality constraints… just check in the *if* statement!

Nested Loop Join (NLJ)

```
Compute R ⋈ S on A:
  for r in R:
    for s in S:
      if r[A] == s[A]:
        yield (r,s)
```

What would ***OUT*** be if our join condition is trivial (*if TRUE)?*

OUT could be bigger than P(R)*P(S)… but usually not that bad

Cost:

P(R) + T(R)*P(S) + OUT

1. Loop over the tuples in R
2. For every tuple in R, loop over all the tuples in S
3. Check against join conditions
4. **Write out (to page, then when page full, to disk)**

Nested Loop Join (NLJ)

Compute $R \bowtie S$ *on* A:

```
for r in R:
  for s in S:
    if r[A] == s[A]:
      yield (r,s)
```

Cost:

P(R) + T(R)*P(S) + OUT

What if R ("outer") and S ("inner") switched?

P(***S***) + T(***S***)*P(***R***) + OUT

Outer vs. inner selection makes a huge difference- DBMS needs to know which relation is smaller!

Block Nested Loop Join (IO-Aware variant)

Block Nested Loop Join (BNLJ)

Given ***B+1*** pages of memory

```
Compute R ⋈ S on A:
  for each B-1 pages pr of R:
    for page ps of S:
      for each tuple r in pr:
        for each tuple s in ps:
          if r[A] == s[A]:
            yield (r,s)
```

Cost:

$P(R)$

1. **Load in B-1 pages of R at a time (leaving 1 page each free for S & output)**

Note: There could be some speedup here due to the fact that we're reading in multiple pages sequentially however we'll ignore this here!

Block Nested Loop Join (BNLJ)

```
Compute R ⋈ S on A:
  for each B-1 pages pr of R:
    for page ps of S:
      for each tuple r in pr:
        for each tuple s in ps:
          if r[A] == s[A]:
            yield (r,s)
```

Given ***B+1*** pages of memory

Cost:

$$P(R) + \frac{P(R)}{B-1} P(S)$$

1. Load in B-1 pages of R at a time (leaving 1 page each free for S & output)

2. **For each (B-1)-page segment of R, load each page of S**

Block Nested Loop Join (BNLJ)

```
Compute R ⋈ S on A:
  for each B-1 pages pr of R:
    for page ps of S:
      for each tuple r in pr:
        for each tuple s in ps:
          if r[A] == s[A]:
            yield (r,s)
```

Given ***B+1*** pages of memory

Cost:

$$P(R) + \frac{P(R)}{B-1} P(S)$$

1. Load in B-1 pages of R at a time (leaving 1 page each free for S & output)
2. For each (B-1)-page segment of R, load each page of S
3. **Check against the join conditions**

BNLJ can also handle non-equality constraints

Block Nested Loop Join (BNLJ)

```
Compute R ⋈ S on A:
  for each B-1 pages pr of R:
    for page ps of S:
      for each tuple r in pr:
        for each tuple s in ps:
          if r[A] == s[A]:
            yield (r,s)
```

Given ***B+1*** pages of memory

Cost:

$$P(R) + \frac{P(R)}{B-1} P(S) + \text{OUT}$$

1. Load in B-1 pages of R at a time (leaving 1 page each free for S & output)
2. For each (B-1)-page segment of R, load each page of S
3. Check against the join conditions
4. **Write out**

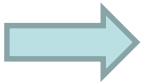
Again, ***OUT*** could be bigger than P(R)*P(S)… but usually not that bad

BNLJ vs. NLJ: Benefits of IO Aware

- In BNLJ, by loading larger chunks of R, we minimize the number of full *disk reads* of S
  - We only read all of S from disk for ***every (B-1)-page segment of R***!
  - Still the full cross-product, but more done only *in memory*

NLJ

P(R) + T(R)*P(S) + OUT



BNLJ

$P(R) + \frac{P(R)}{B-1} P(S) +$ OUT

BNLJ is faster by roughly $\frac{(B-1)T(R)}{P(R)}$!

BNLJ vs. NLJ: Benefits of IO Aware

- Example:
  - R: 500 pages
  - S: 1000 pages
  - 100 tuples / page
  - We have 12 pages of memory (B = 11)

Ignoring OUT here…

- NLJ: Cost = 500 + **50,000*1000** = **50 Million IOs ~= 140 hours**
- BNLJ: Cost = 500 + $\frac{500*1000}{10}$ = **50 *Thousand* IOs ~= 0.14 hours**

A very real difference from a small change (IO-awareness) in the algorithm!

Nested Loops Join cont.

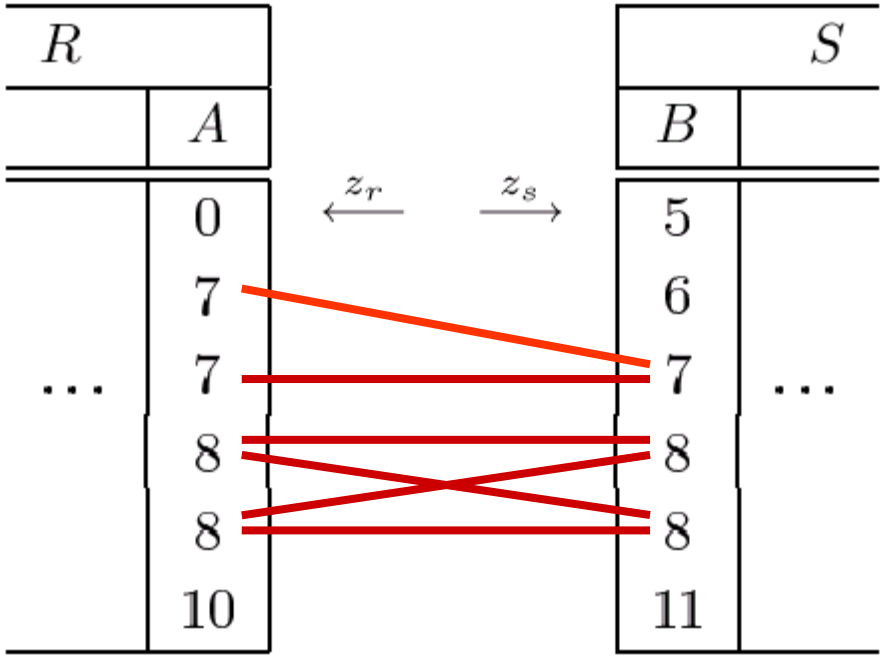
- **index nested loops join** …
- **temporary index nested loops join** …

Nested Loops Join

- Pros: any join condition
- Cons: expensive
  - Better, if
    - Tables fit in memory
    - Smaller table fits in memory (-> inner table)

Sort-Merge-Join

- Sort relations by join attribute,
  Interleaved linear scan



Sort-merge join

```
function sortMerge(relation left, relation
  right, attribute a)
      var relation output
      var list left_sorted := sort(left, a)
      var list right_sorted := sort(right, a)
      var left_key
      var right_key
      var set left_subset
      var set right_subset
…
```

Sort-merge join 2

```
…
advance(left_subset, left_sorted, left_key, a)
     advance(right_subset, right_sorted, right_key, a)
     while not empty(left_subset) and not empty(right_subset)
         if left_key = right_key
             add cross product of left_subset
                     and right_subset to output
             advance(left_subset, left_sorted, left_key, a)
             advance(right_subset,right_sorted, right_key, a)
         else if left_key < right_key
            advance(left_subset, left_sorted, left_key, a)
         else // left_key > right_key
            advance(right_subset, right_sorted, right_key, a)
     return output
…
```

Sort-merge join 3

```
...
function advance(subset, sorted, key, a)
      key = sorted[1].a
      subset = emptySet
      while not empty(sorted) and
                                sorted[1].a = key
          insert(subset, sorted[1])
          remove first element from sorted
```

Sort-merge join

- Join operators
  - Also <, <=, >, >=

Simple Hash Join (1)

- Partition relation R in R_1, R_2, ..., R_p with Hash-function h so that for all tuples $r \in R_i$ h($r.A$) $\in H_i$.
- Scan relation S, apply for each tuple $s \in S$ the hash function h. Is $h(s.B)$ in H_i, search there for fitting r

```
repeat
    begin
    while memory not exhausted do
      insert next(R) into partition[h(next(R).A)];
    foreach s ∈ S do
      foreach r ∈ partition[h(s.B)] do
          if r.A = s.B Res := Res ∪ (r ◦ s)
    end
until R exhausted
```

Build-phase

Probe-phase

Hash Join

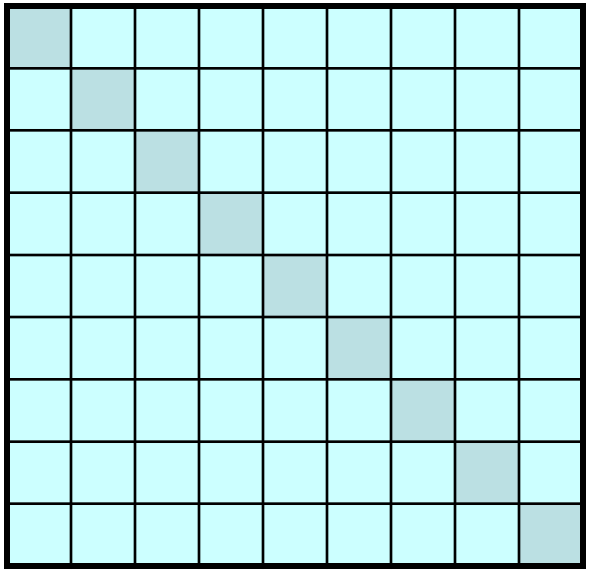
- Size of hash-table > available RAM
  - Multiple scans! (Very) slow!
  - (Small increase of data can cause large increase in run time!)

- (Grace hash join)
  - Partitioning R and S via a hash function
  - Join only within partitions
  - avoids multiple rescanning of entire *S* relation

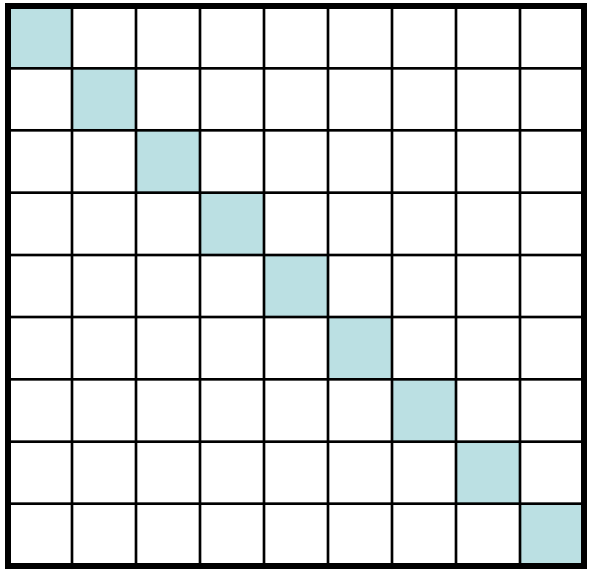
Hash Join

- Pros
  - quick (if memory sufficient)!
- Cons
  - Memory requirement
  - Join operator: only „=“

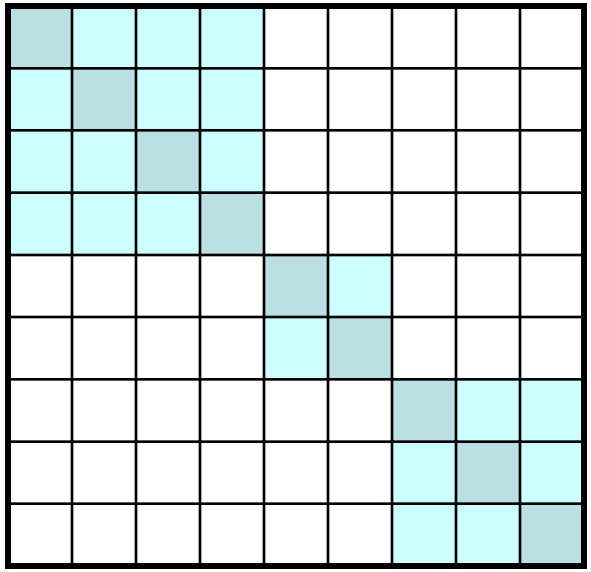
Resource requirement, complexity (comparison step)



Nested-Loop



Merge



Hash

Element comparisons

Successful element comparisons

Comparison

- Size and schema of relations in join
- Available indices
- Other operations (presorted intermediate results)
- Available memory
- Join-type (natural join, theta-join, equijoin; outer join?)

Cost based query optimization

37

Query execution

- *SQL: very high level, declarative*
  - *What* to retrieve, not *how*!
- SQL query is translated by the *query processor* into a low level program – the *execution plan*
- An execution plan is a program that can be executed to get the answer to the query.

Key Idea: Algebraic Optimization

- N = ((n*5)+(n*2)+0)/1
- Algebraic laws:
  1. (+) identity: x+0=x
  2. (/) identity: x/1=x
  3. (*) distributes: (y*x)+(z*x)=(y+z)*x
  4. (*) commutes: y*x=x*y
- Rules 1, 3, 4,2:
  - N=(5+2)*n

Relational algebra Equivalences

1. Selection

$$\sigma_{c_1 \wedge c_2 \wedge \ldots \wedge c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(\ldots(\sigma_{c_n}(R)) \ldots))$$

2. σ is commutative

$$\sigma_{c_1}(\sigma_{c_2}((R)) \equiv \sigma_{c_2}(\sigma_{c_1}((R))$$

3. π -cascades: If $L_1 \subseteq L_2 \subseteq \ldots \subseteq L_n$, then

$$\pi_{L_1}(\pi_{L_2}(\ldots(\pi_{L_n}(R)) \ldots)) \equiv \pi_{L_1}(R)$$

4. Changing σ and π

   If the selection only refers to the projected attributes $A_1, \ldots, A_n$, selection and projection can be exchanged

$$\sigma_c(\pi_{A_1, \ldots, A_n}(R)) \equiv \pi_{A_1, \ldots, A_n}(\sigma_c(R))$$

Equivalences 2

5. ×, ⋈, ∪ and ∩ are commutative

6. A Cartesic product, followed by a selection referring to both operands can be replaced by a join.

$$\sigma_c(R \times S) \equiv R \bowtie_c S$$

....

Equivalences…

$\sigma_{c_1 \wedge c_2 \wedge \ldots \wedge c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(\ldots(\sigma_{c_n}(R)) \ldots))$

$\sigma_{c_1}(\sigma_{c_2}((R)) \equiv \sigma_{c_2}(\sigma_{c_1}((R))$

$\pi_{L_1}(\pi_{L_2}(\ldots(\pi_{L_n}(R)) \ldots)) \equiv \pi_{L_1}(R)$

$\sigma_c(\pi_{A_1, \ldots, A_n}(R)) \equiv \pi_{A_1, \ldots, A_n}(\sigma_c(R))$

$R \Phi S \equiv S \Phi R$ ($\bowtie$, □, $\cup$, $\cap$)

$\sigma_c(R \square S) \equiv R \bowtie_c S$

$\sigma_c(R \Phi S) \equiv \sigma_c(R) \Phi S$ ($\bowtie$, □)

$\sigma_c(R \Phi S) \equiv \sigma_{c_1}(R) \Phi \sigma_{c_2}(S)$ ($\bowtie$, □)

$\pi_L(R \bowtie_c S) \equiv (\pi_{A_1, \ldots, A_n}(R)) \bowtie_c (\pi_{B_1, \ldots, B_n}(S))$

$\pi_L(R \bowtie_c S) \equiv \pi_L((\pi_{A_1, \ldots, A_n, A_1', \ldots, A_n'}(R)) \bowtie_c (\pi_{B_1, \ldots, B_n, B_1', \ldots, B_n'}(S)))$

$(R \Phi S) \Phi T \equiv R \Phi (S \Phi T)$ ($\bowtie$, □, $\cup$, $\cap$)

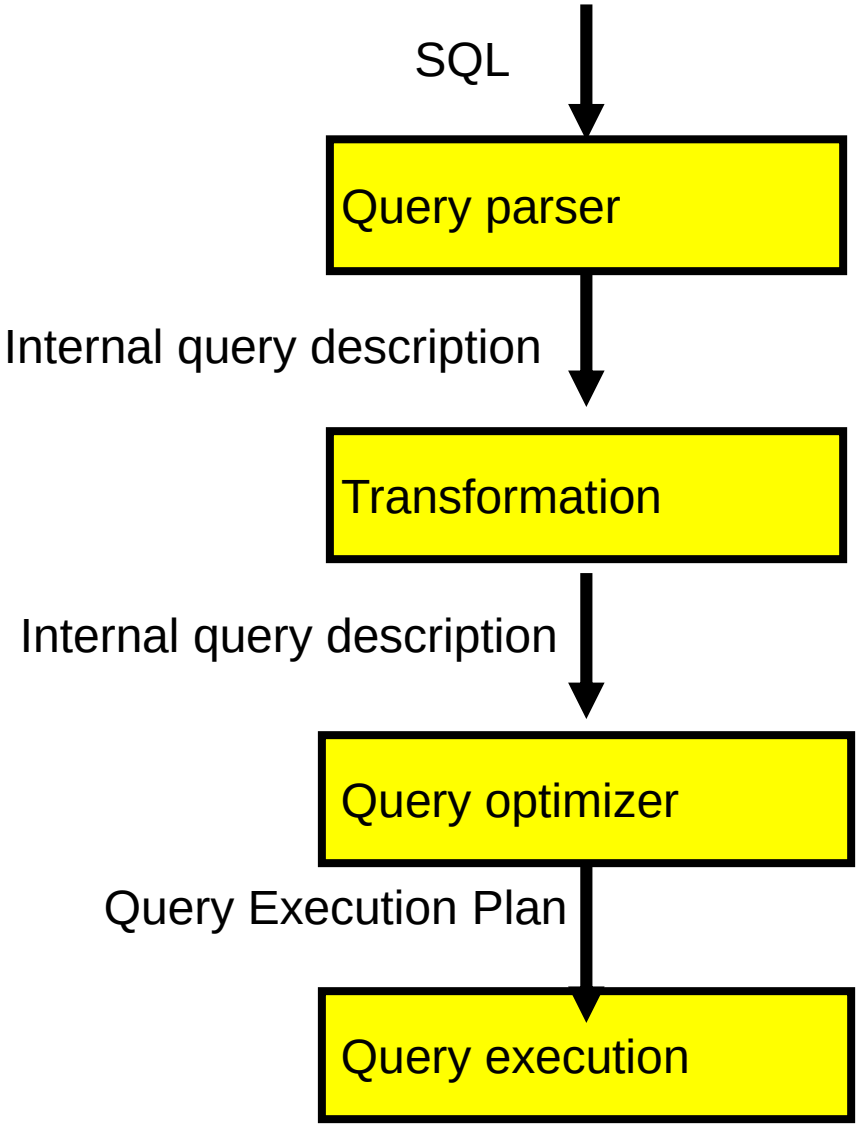
$\sigma_c(R \Phi S) \equiv (\sigma_c(R)) \Phi (\sigma_c(S))$ ($\cup$, $\cap$, –)

$\pi_L(R \Phi S) \equiv (\pi_L(R)) \Phi (\pi_L(S))$ ($\cup$, $\cap$, –)

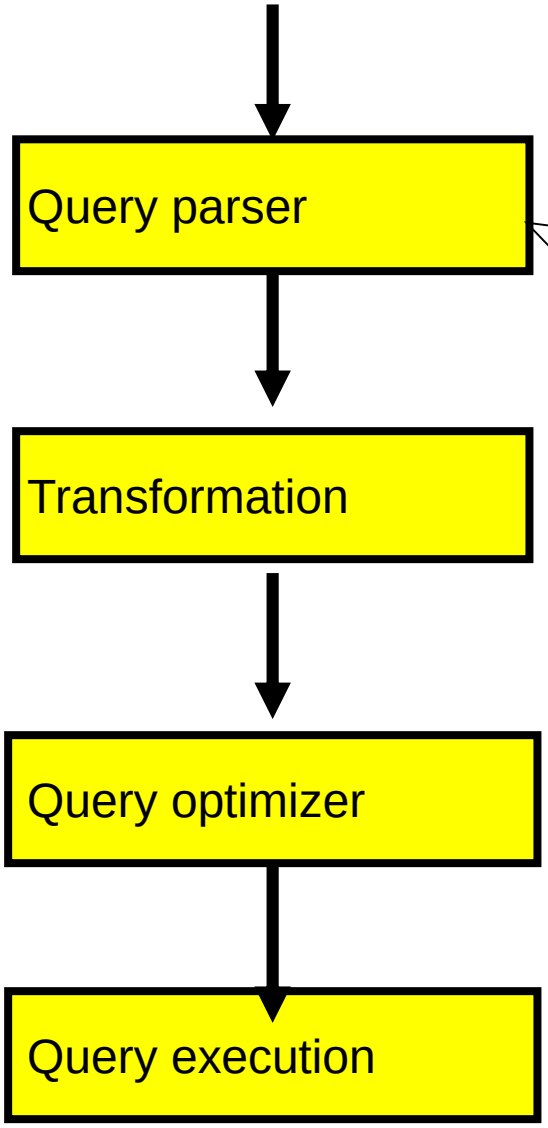
Query optimisation

- One SQL query – many (!) different execution plans, execution alternatives
  - Index – using, not using
    - tendence: selective query – using
    - Multiple indices, which index to use?
  - Join execution?
    - Size of tables, available memory…
  - Join order ?
- Dramatically different costs!
- Query optimizer: choosing a relatively good execution plan

Query execution

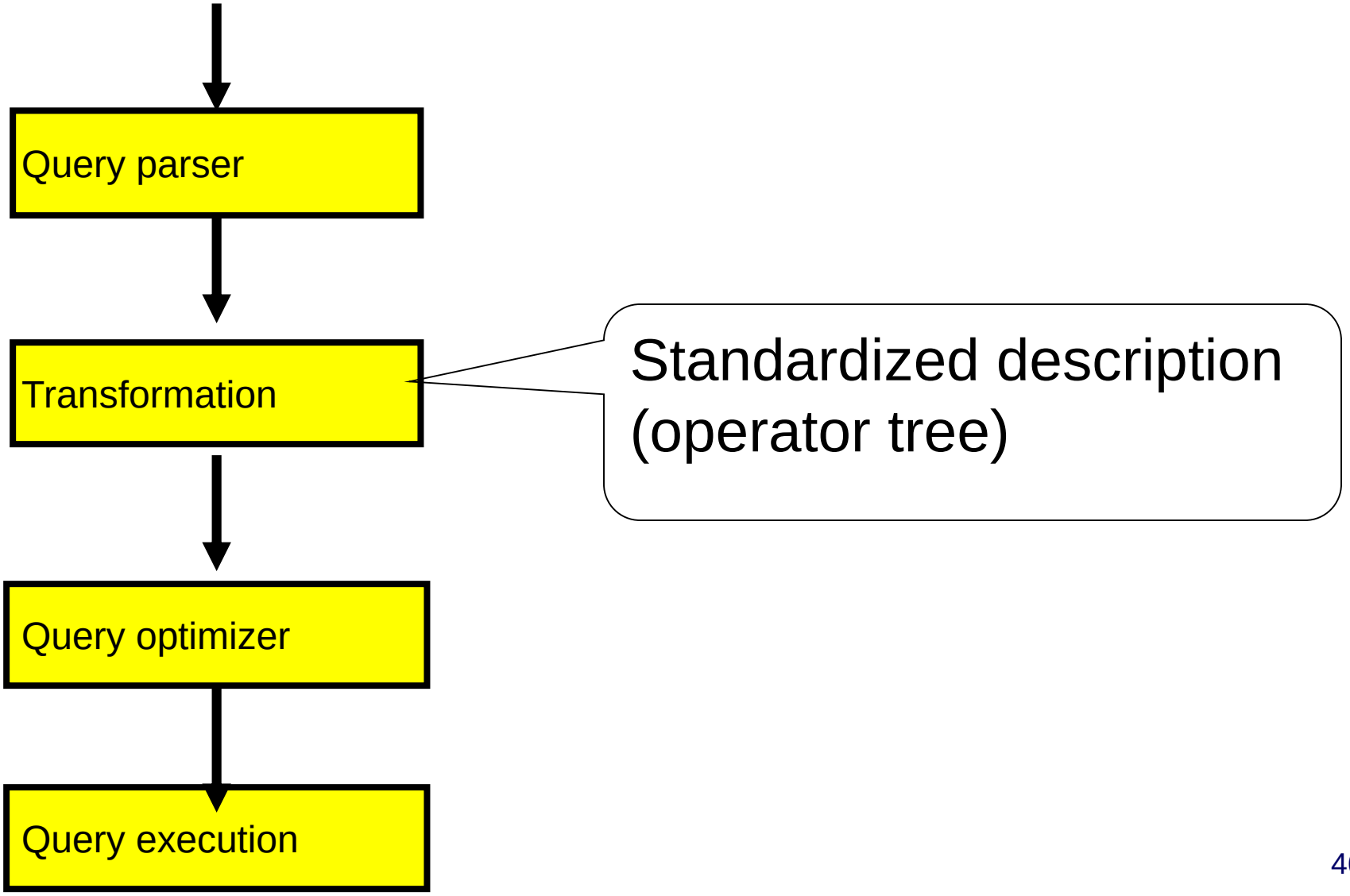


Query execution (detailed)

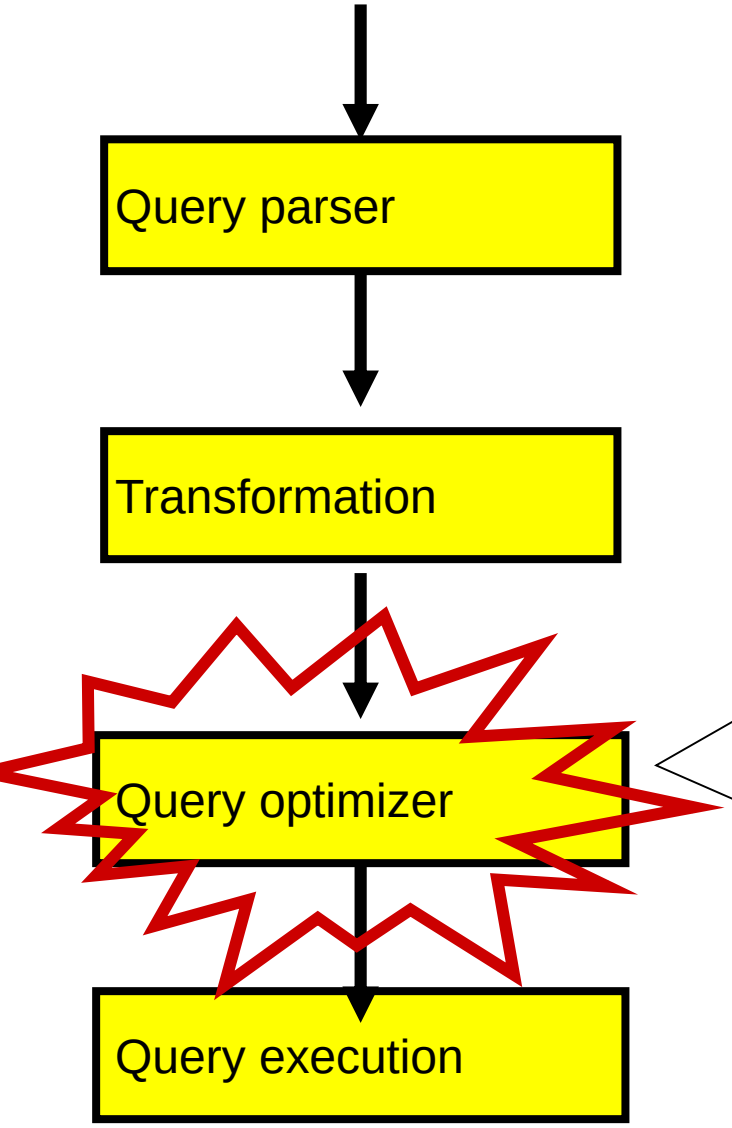


Parsing
Semantic check (tables, attributes – Data Dictionary)
View expansion

Query execution (detailed)



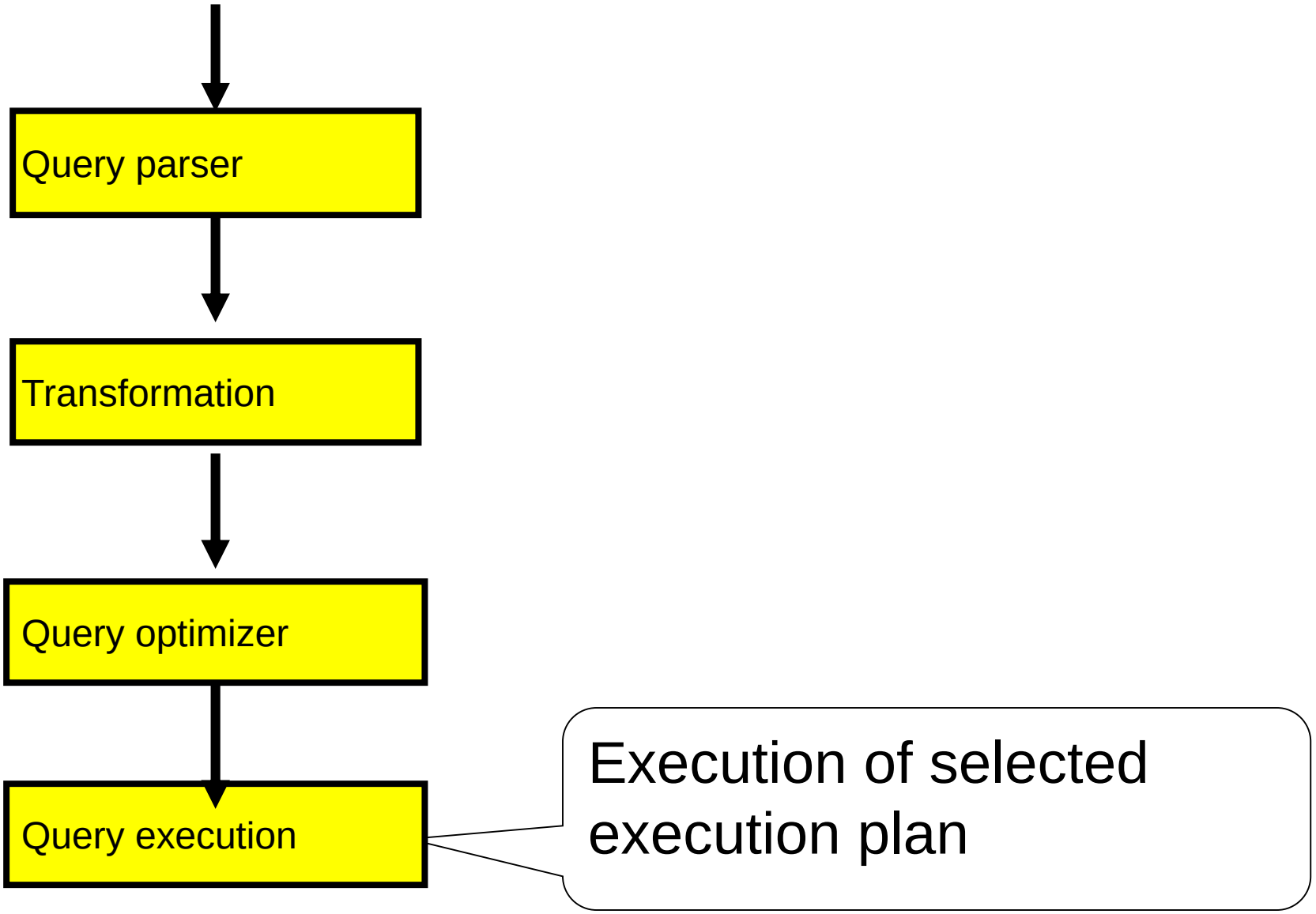
Query execution (detailed)



Optimisation (cost-based):

- Setting up execution plans
- Estimating their costs
- Selecting a cheap execution plan

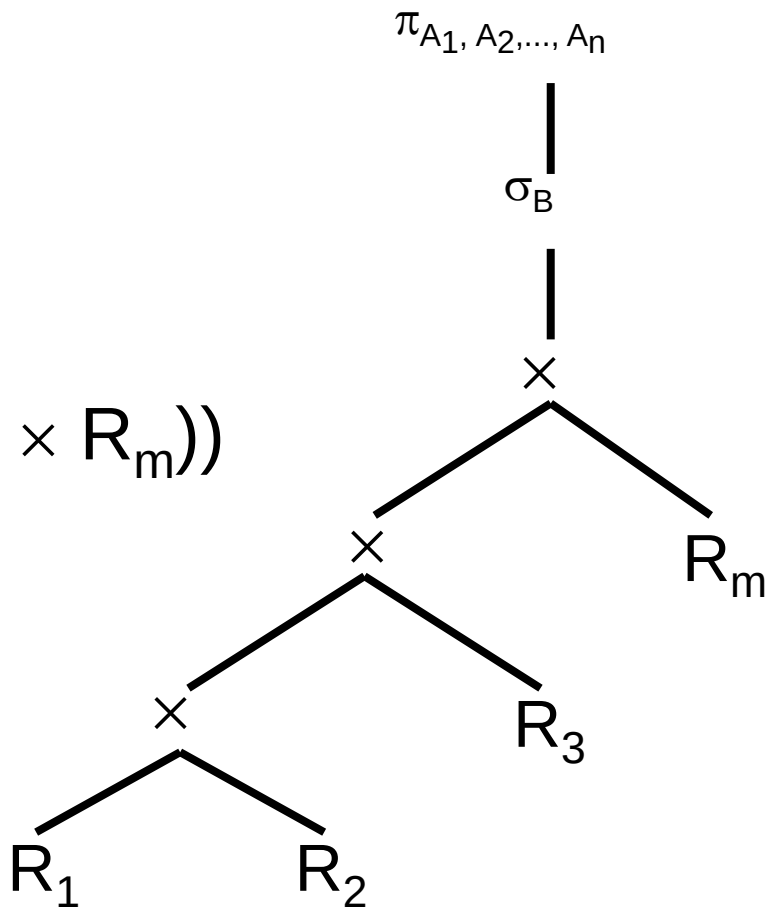
Query execution (detailed)



Transformation, operator tree

```
select   A1, A2,..., An
from R1, R2,..., Rm
where    B
```

$\Rightarrow \pi_{A_1, A_2, \ldots, A_n} (\sigma_B (R_1 \times R_2 \times \ldots \times R_m))$

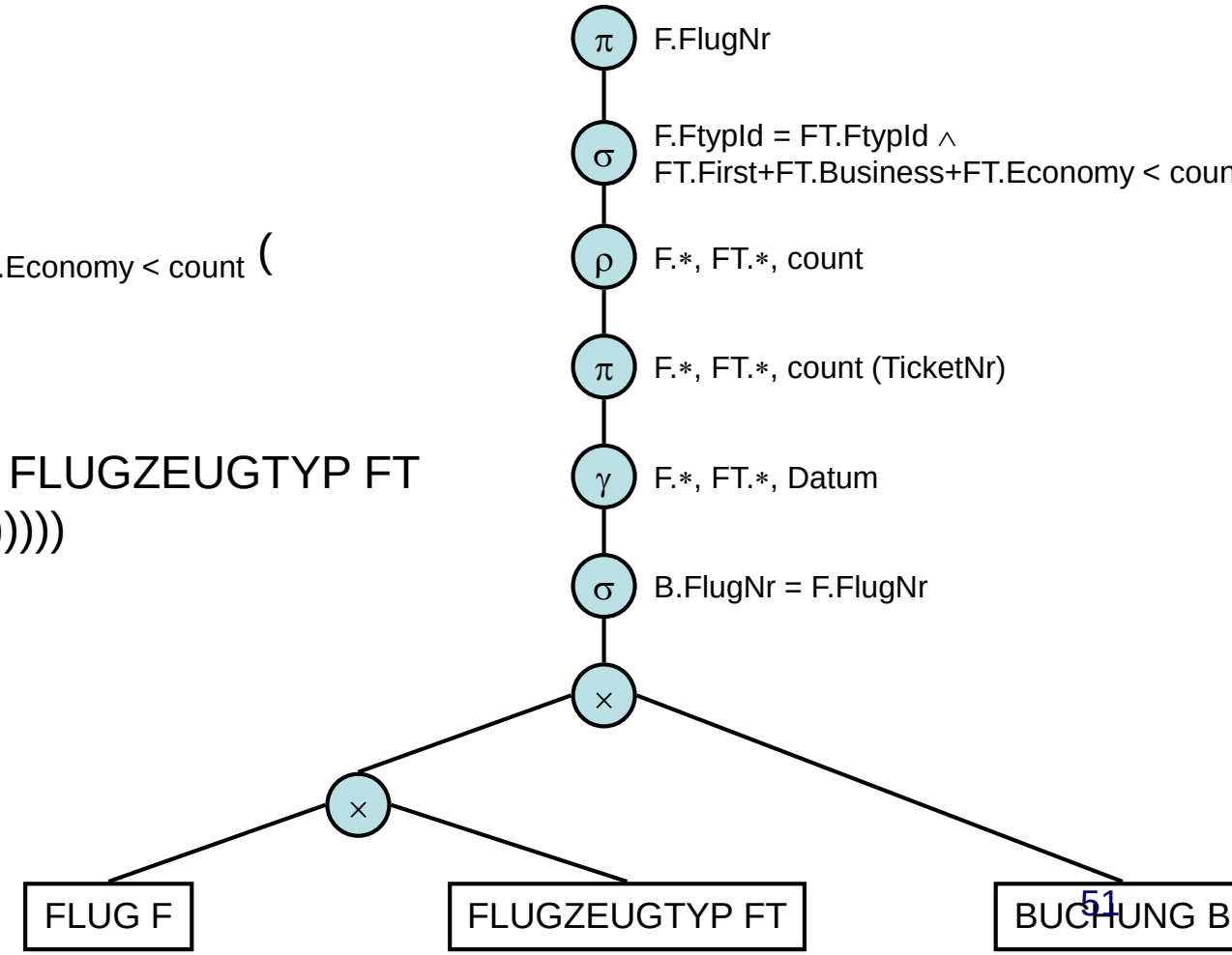


Example

```
select FlugNr
from   (select F.*, FT.*, count (TicketNr)
        from   FLUG F, FLUGZEUGTYP FT,
               BUCHUNG B
        where  B.FlugNr = F.FlugNr
        group by              F.*, FT.*, Datum)
       as DFT(F.*,FT.*,count)
where  F.FtypId = FT.FtypId
and    FT.First+FT.Business+FT.Economy
                           < DFT.count
```

Example

$\pi_{F.FlugNr}$ (
$\sigma_{F.FtypId =}$
FT.FtypId $\wedge$ FT.First+FT.Business+FT.Economy < count (
$\rho_{F.*,FT.*,count}$ (
$\pi_{F.*,\ FT.*,\ count\ (TicketNr)}$ (
$\gamma_{F.*,\ FT.*,\ Datum}$ (
$\sigma_{B.FlugNr = F.FlugNr}$ (FLUG F $\times$ FLUGZEUGTYP FT
BUCHUNG B))))))



Basic idea

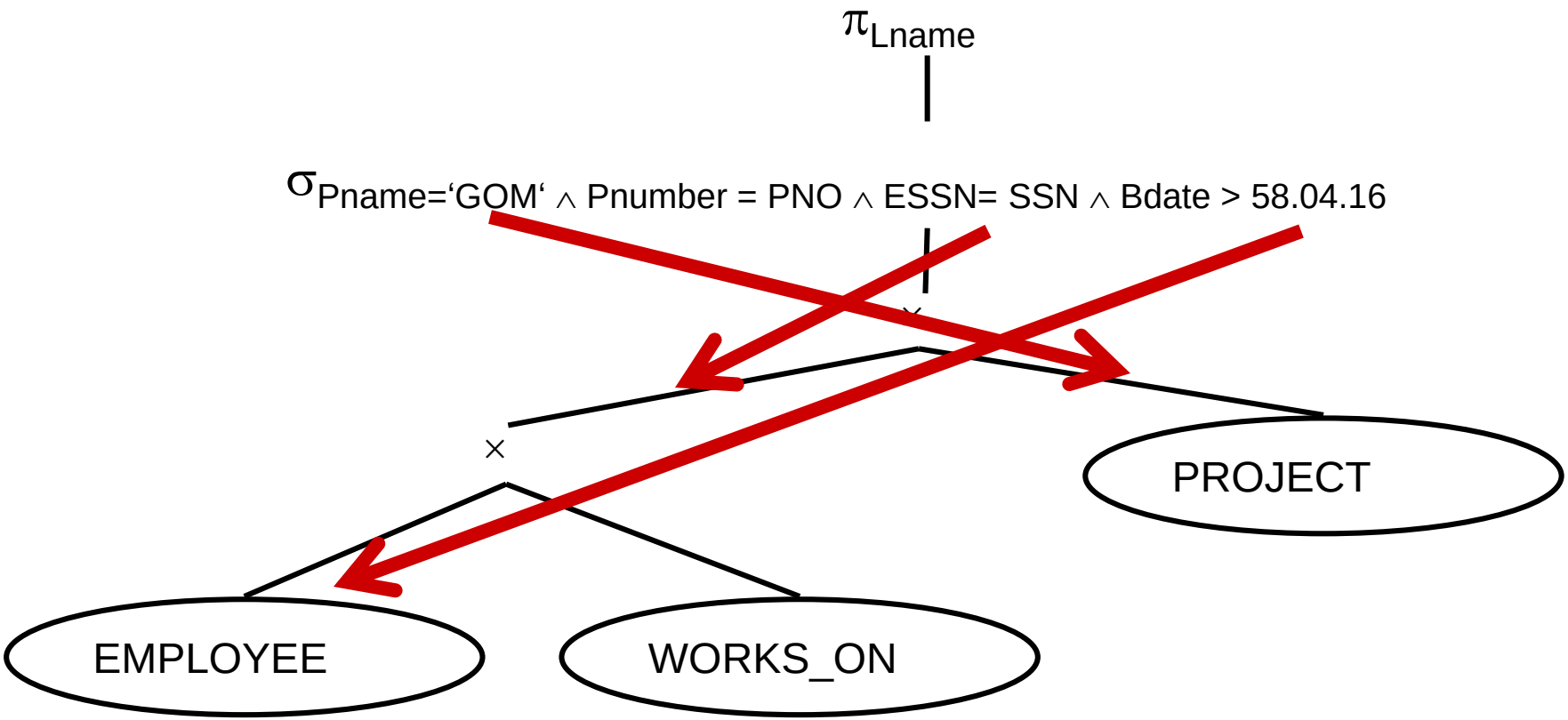
- Some transformations always reasonable
- Some transformations depend on data
  - Optimizing the average case
  - Keep intermediate results as small as possible

- Executing σ, π early, $\bowtie$, ⍰, $\cup$,… late, as
  - σ and π reduces the volume of data,
  - $\bowtie$, … result often in large intermediate results.

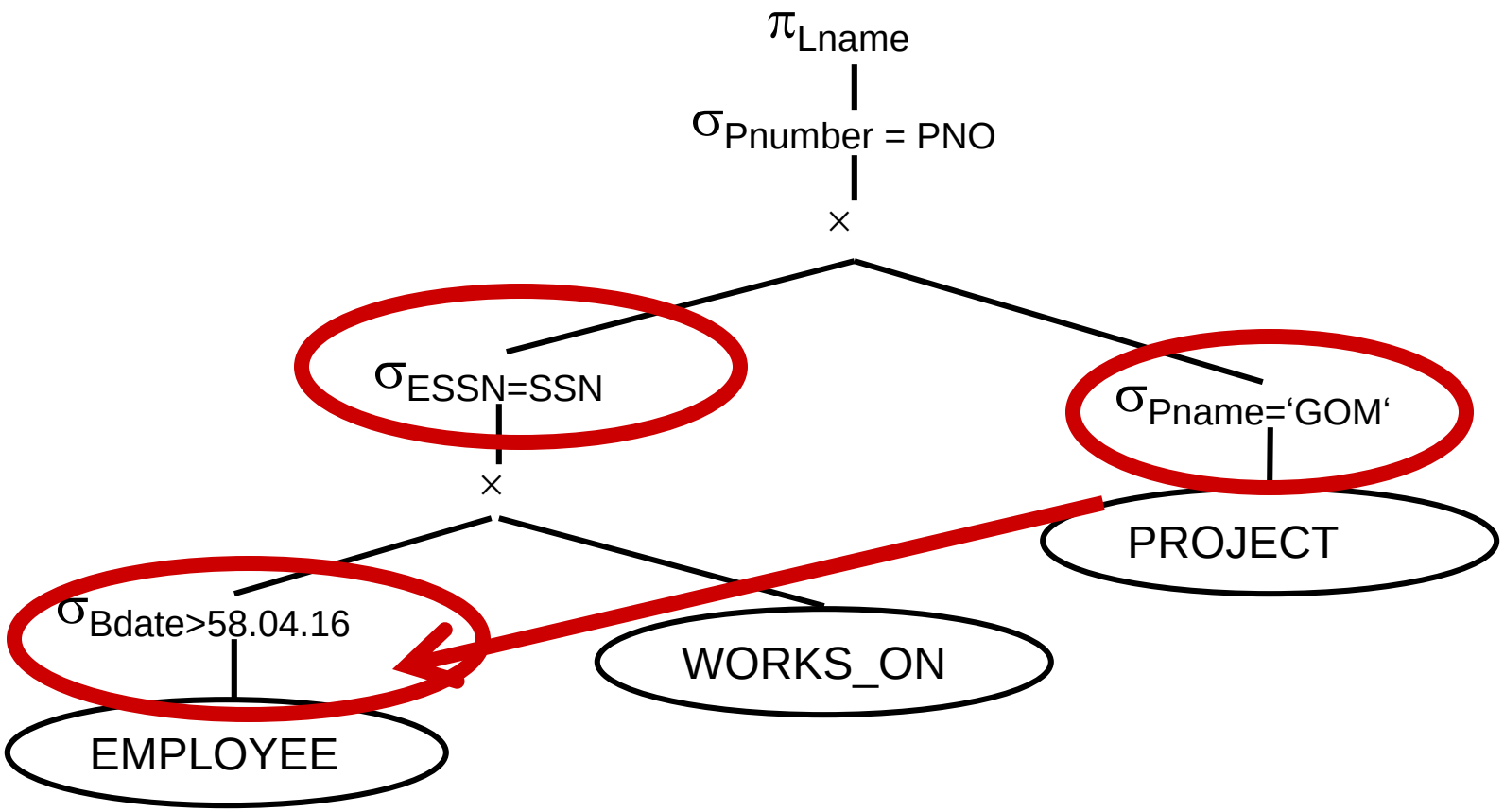
Example

```
select Lname
from   Employee, WorksOn,
Project
where  Pname = 'GOM'
and    Pnumber = PNO
and    ESSN = SSN
and    Bdate > 58.04.16
```

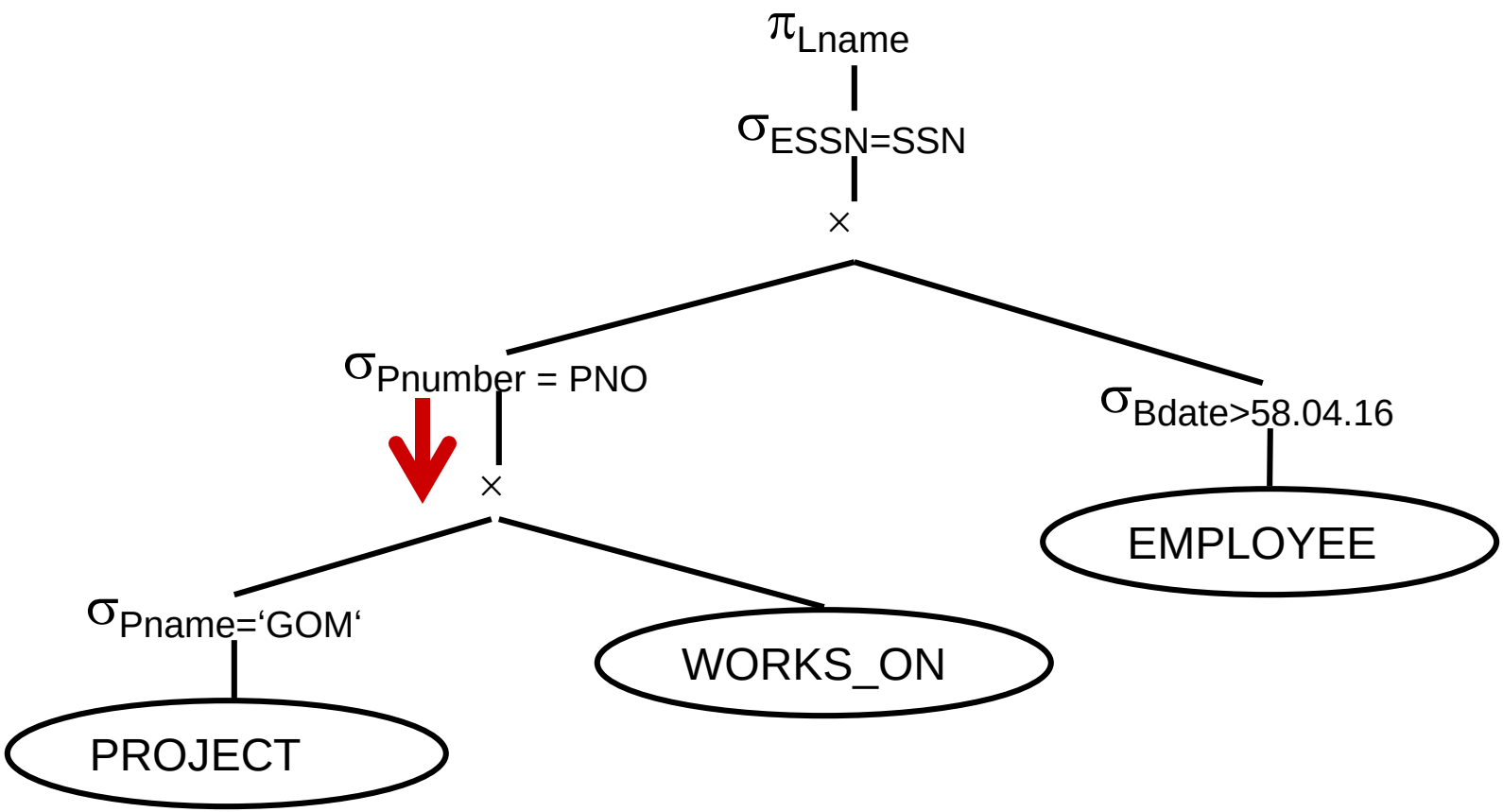
(Select the lastname of an employee born after 16.04.58 and working on the project „GOM”)



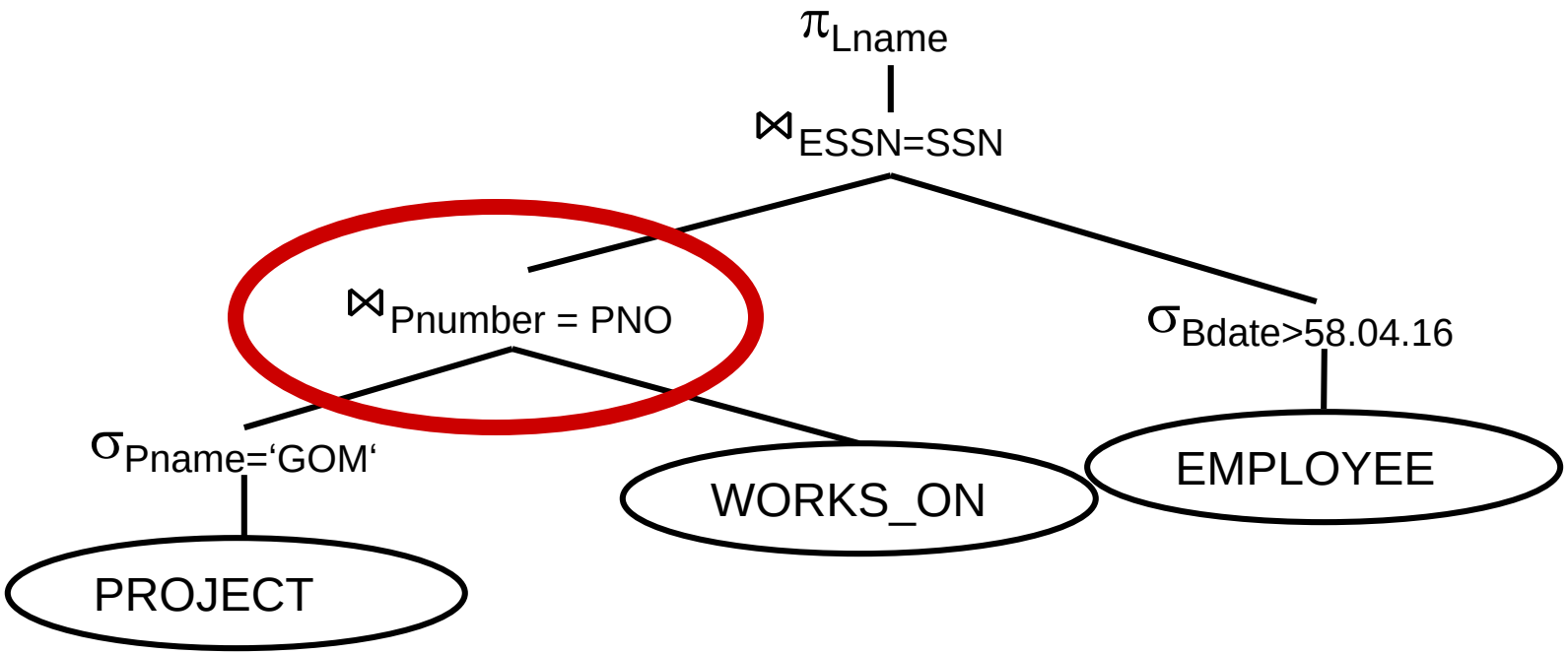
Selection as early, as possible



Restrictive joins early

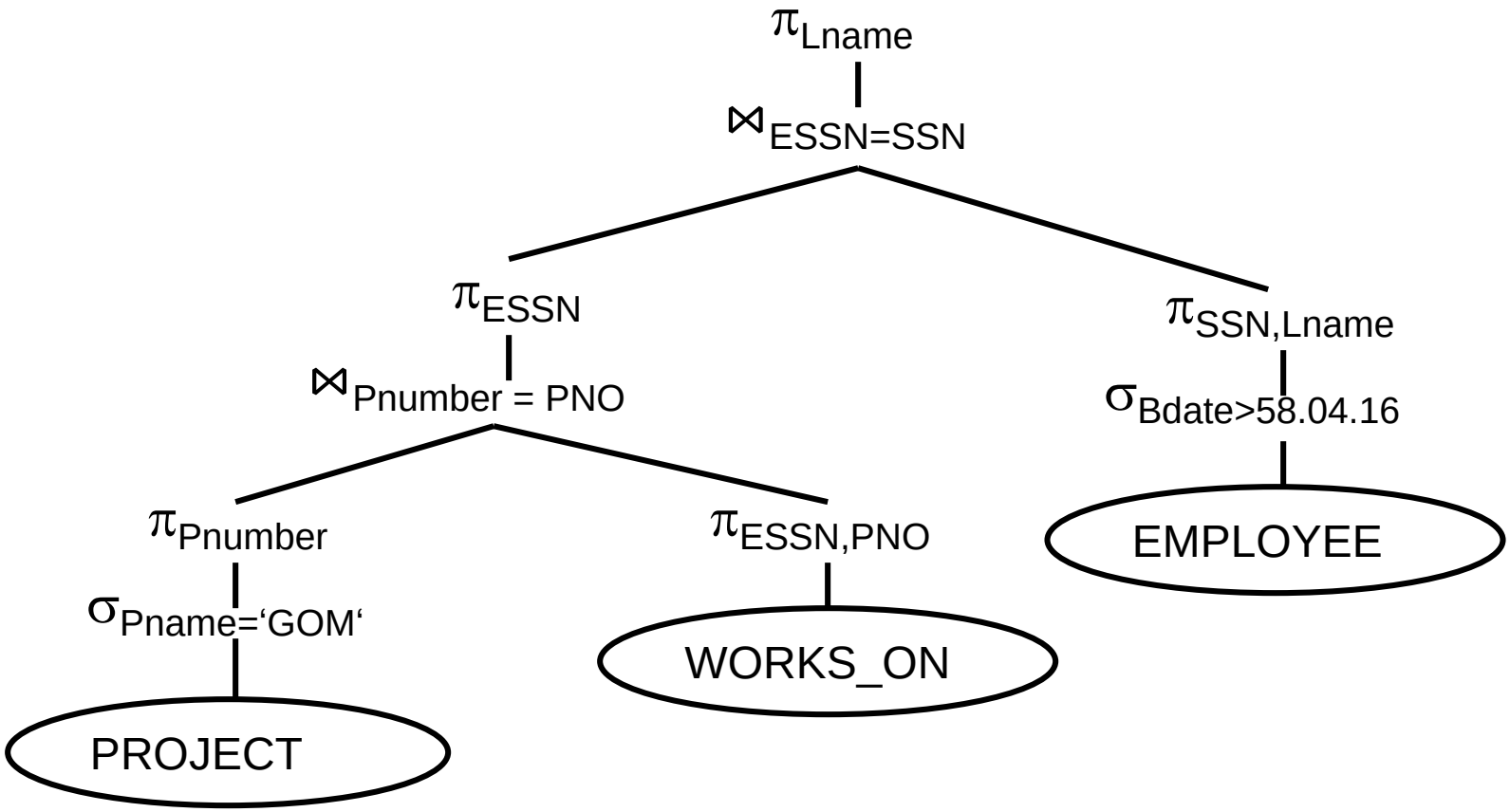


Cross product and selection => join



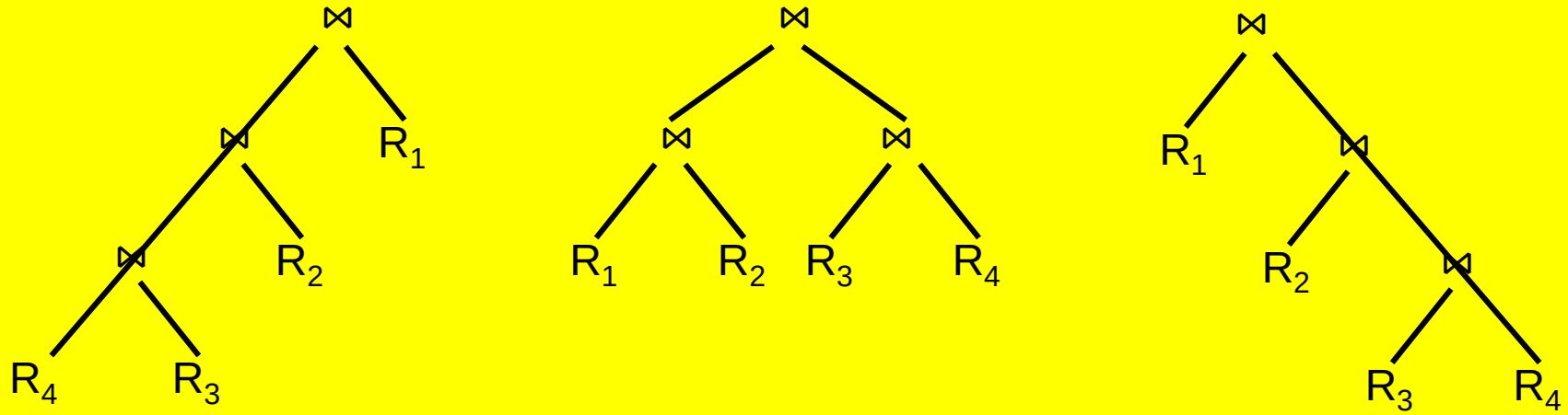
Projections as early, as possible

(attributes for result and intermediate results kept)


πLname
⋈ESSN=SSN
πESSN
πSSN,Lname
⋈Pnumber = PNO
σBdate>58.04.16
πPnumber
πESSN,PNO
EMPLOYEE
σPname='GOM'
WORKS_ON
PROJECT

Join-order

- Many joins
- Joins expensive

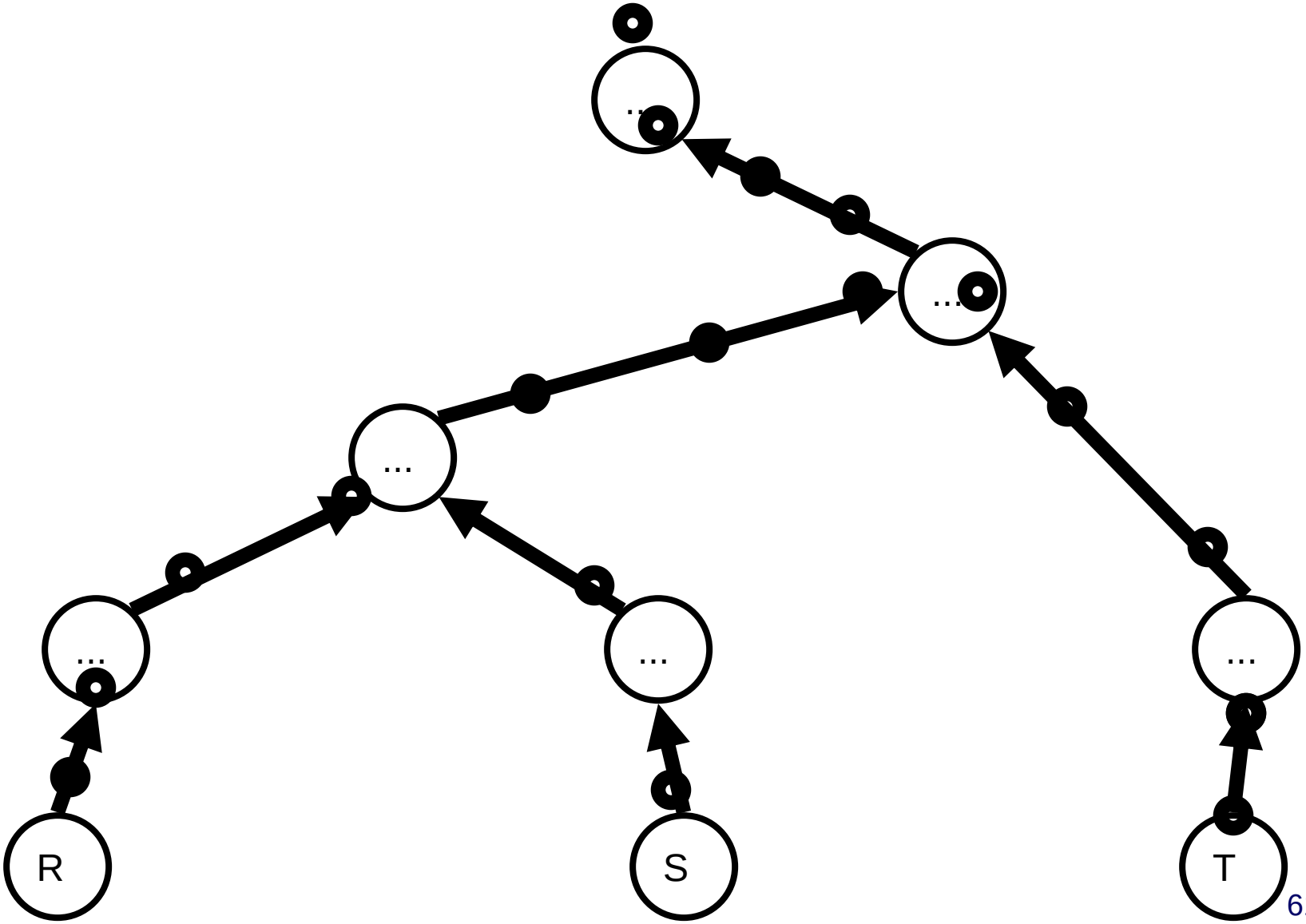


Left oriented join trees, greedy search...

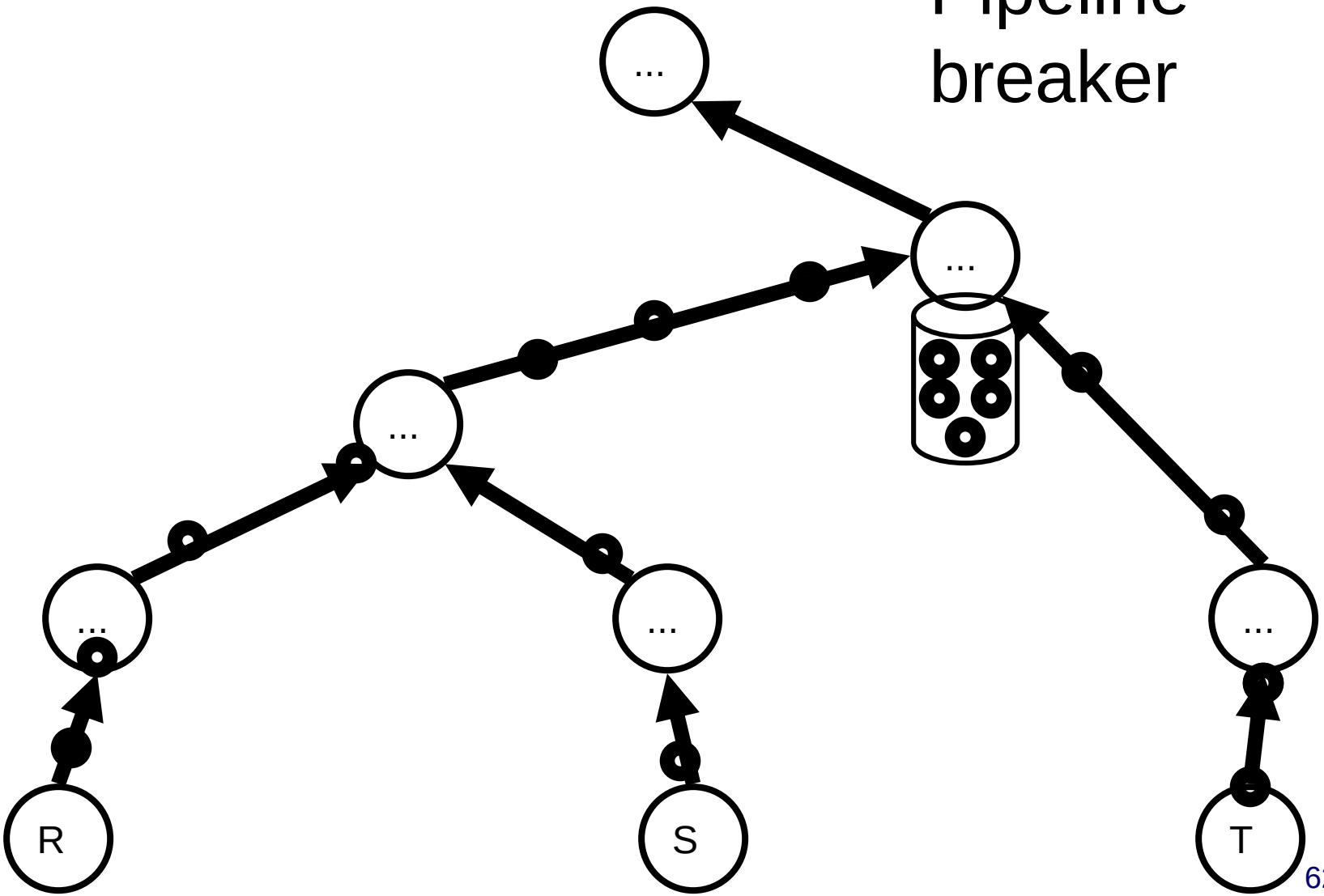
Execution mode

- Full calculation
  - Node fully calculated before next operator
- Pipeline
  - Tuple calculated at one node sent immediately to next node

Execution modes (2)



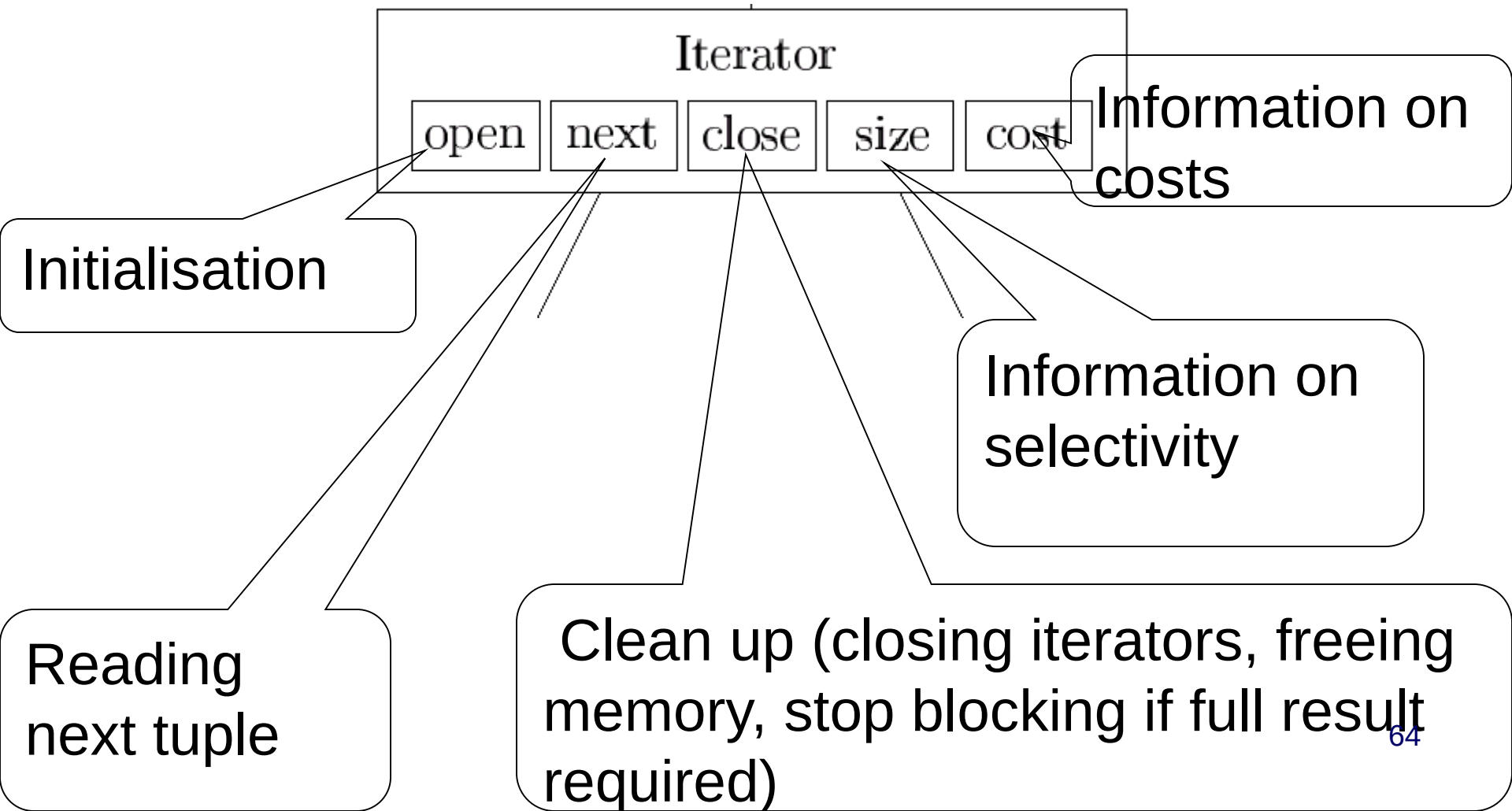
Pipeline breaker

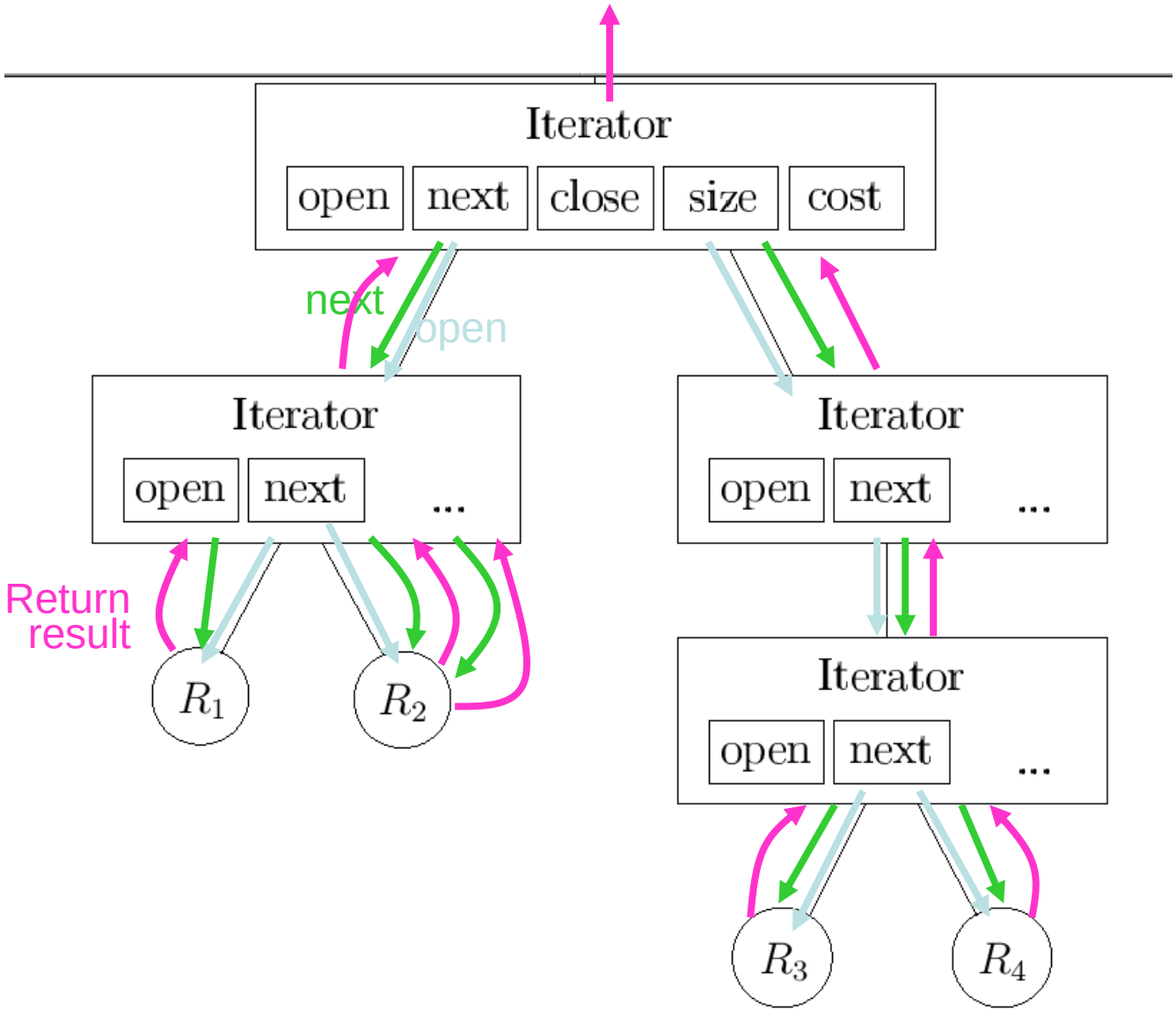


Pipeline-Breaker

- Unary operations
  - Sort
  - Duplicate elimination (unique distinct)
  - Aggregation (min, max, sum)
- Binary operations
  - Set difference
- Depending on the implementation
  - Join
  - Union

Unified description of operators

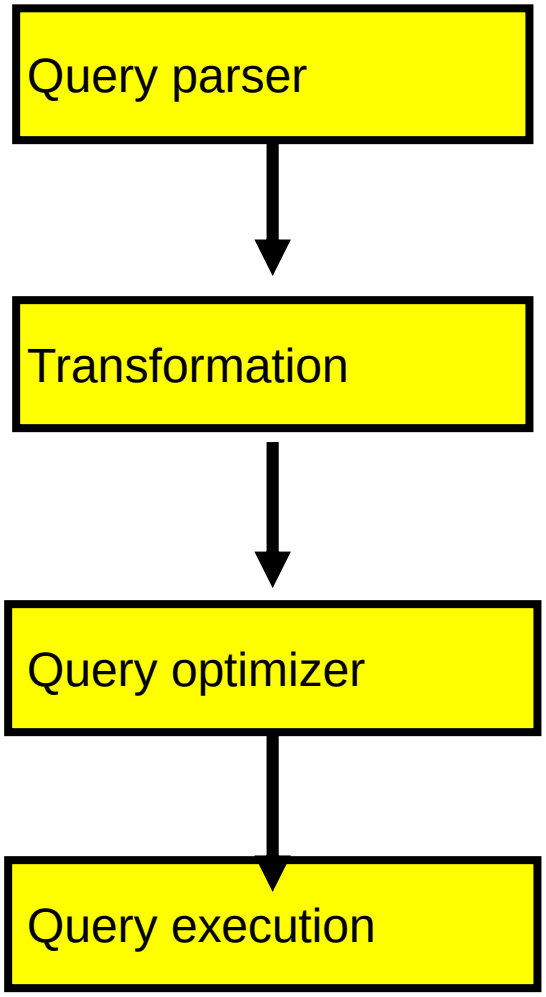



Iterator
open
next
close
size
cost
next
open
Iterator
open
next
...
Iterator
open
next
...
Return
result
R_1
R_2
Iterator
open
next
...
R_3
R_4

TOP-N Query

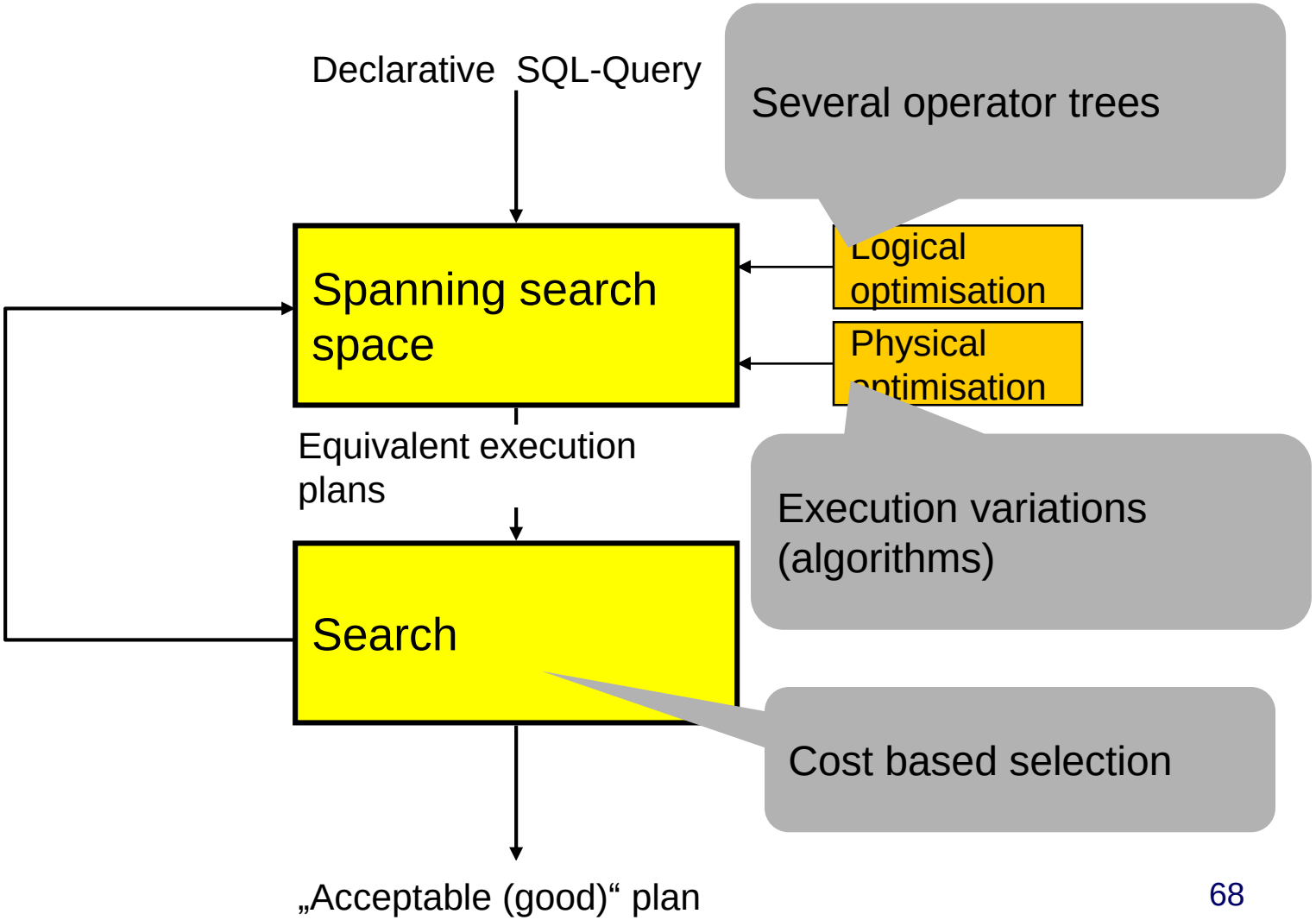
- Only the first N rows of the result are required
- Execution plan optimized specifically for this
  - Hard decisions
  - Soft decisions (internal restart of query possible with a low probability)

Cost based selection



Optimisation (cost-based):

- Setting up execution plans
- Estimating their costs
- Selecting a cheap execution plan


Declarative SQL-Query
Several operator trees
Spanning search space
Logical optimisation
Physical optimisation
Equivalent execution plans
Execution variations (algorithms)
Search
Cost based selection
„Acceptable (good)“ plan

Cost function

- Cost of execution
  - One major factor: disk access, number of blocks
- Size of intermediate result of fundamental importance
  - Estimation on selectivities required
- DBMS calculates and stores different statistics on the data

Oracle

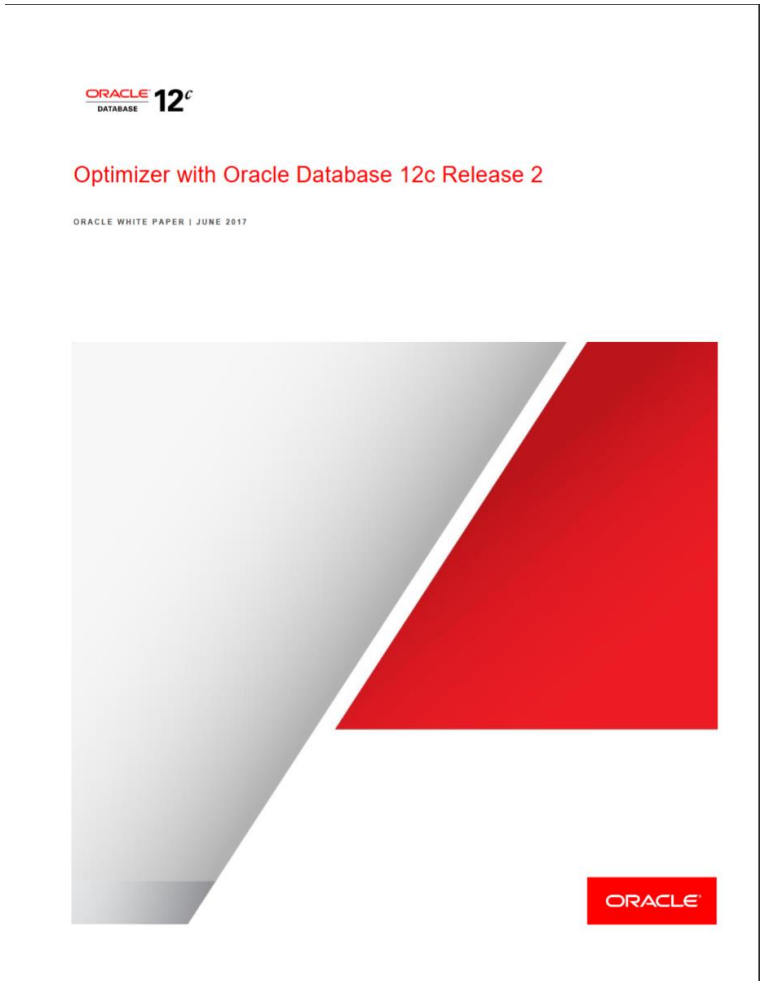
70

Database SQL Tuning Guide (Oracle 12c, Database Administration)

Database SQL Tuning Guide

Contents

Expand All · Collapse All



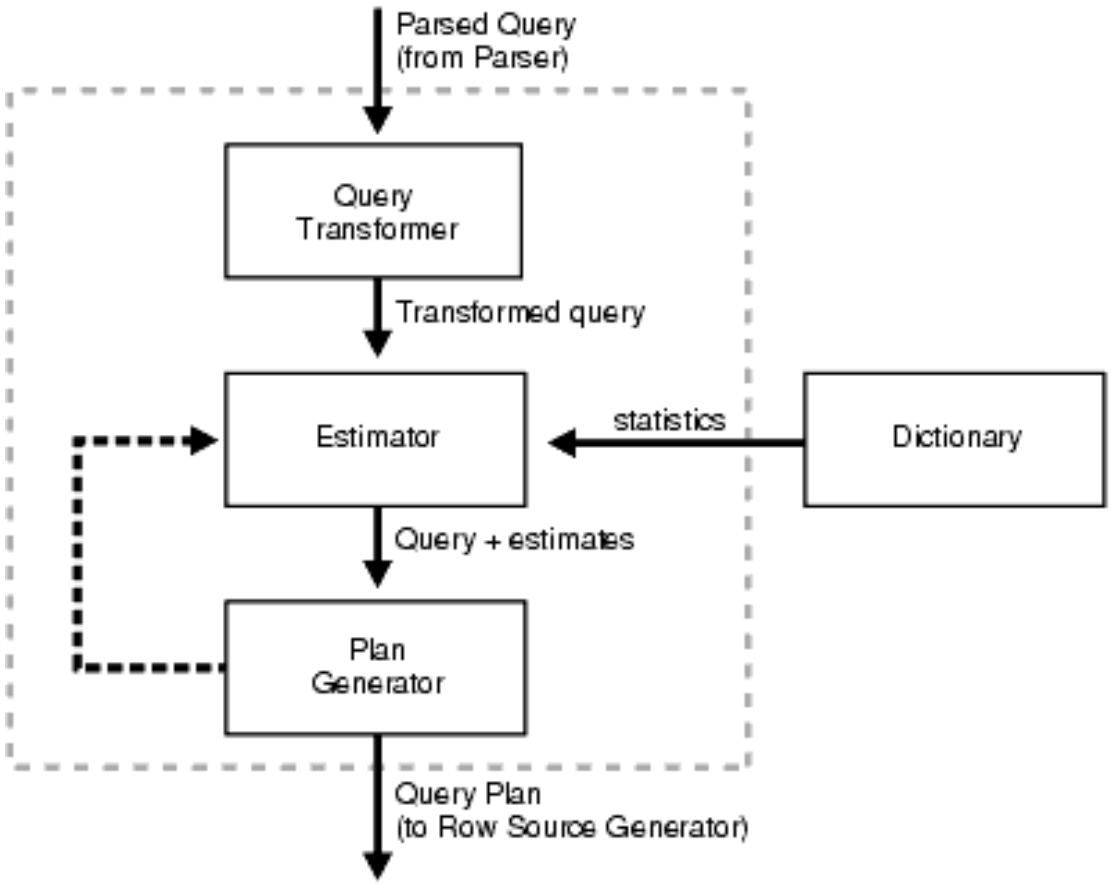
Optimizer Operations

Table 11-1 Optimizer Operations

Operation	Description
Evaluation of expressions and conditions	The optimizer first evaluates expressions and conditions containing constants as fully as possible.
Statement transformation	For complex statements involving, for example, correlated subqueries or views, the optimizer might transform the original statement into an equivalent join statement.
Choice of optimizer goals	The optimizer determines the goal of optimization. See "Choosing an Optimizer Goal".
Choice of access paths	For each table accessed by the statement, the optimizer chooses one or more of the available access paths to obtain table data. See "Overview of Optimizer Access Paths".
Choice of join orders	For a join statement that joins more than two tables, the optimizer chooses which pair of tables is joined first, and then which table is joined to the result, and so on. See "How the Query Optimizer Chooses Execution Plans for Joins".

- Best throughput
- Best response time

Components of the Query Optimizer



Costs

The cost is an estimated value proportional to the expected resource use needed to execute the statement with a particular plan. The optimizer calculates the cost of access paths and join orders based on the estimated computer resources, which includes **I/O, CPU, and memory.**

Statistics in databases

- **Oracle**
  - **TABLES**:
    - **num_rows**,
    - **num_blocks**
    - **avg_row_len**
  - **TAB_COL_STATISTICS**
    - **num_distinct**
    - **num_nulls**
    - **num_buckets**
- **INDEXES**
  - **leaf_blocks**
  - **blevel**

Calculation of statistics

Task of database administrator (expensive task!)

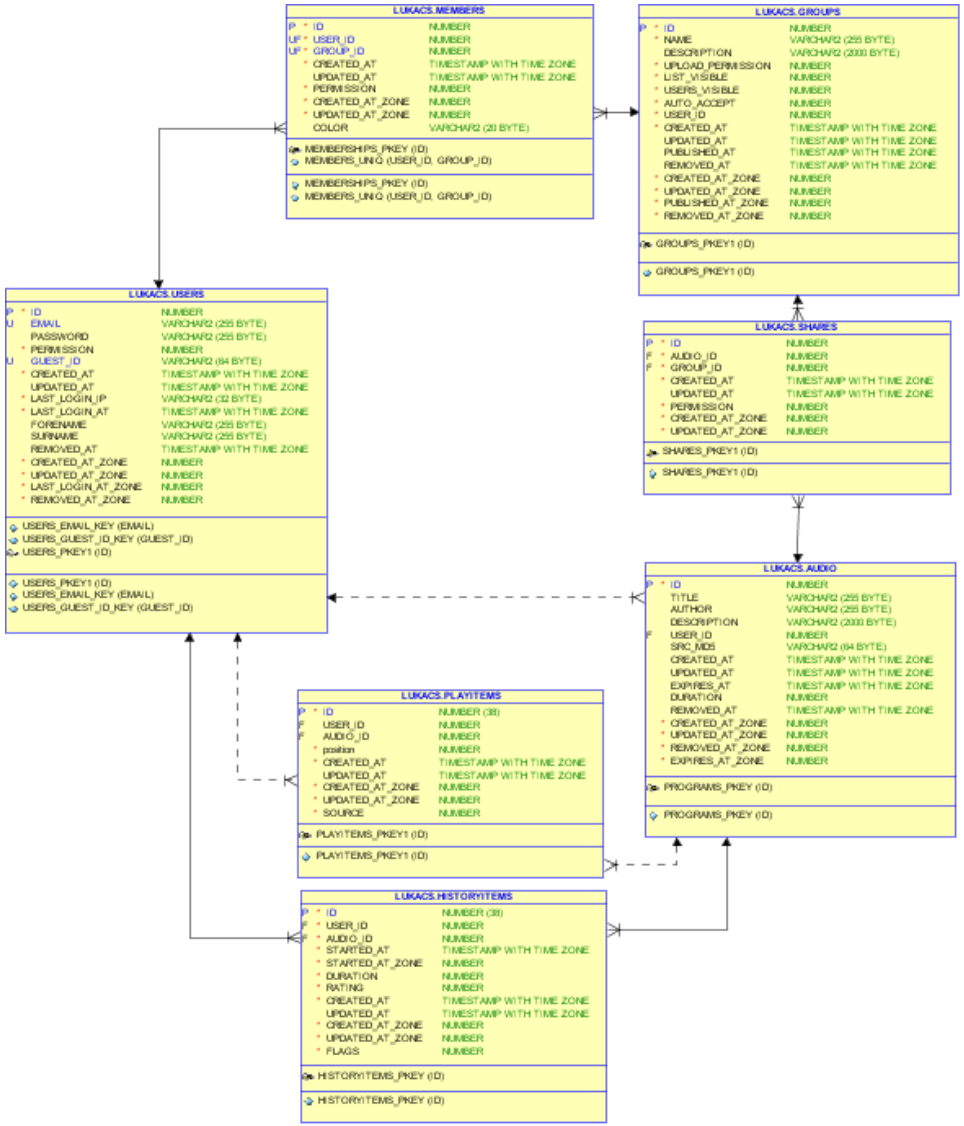
`analyze table` *relation*
`compute statistics for columns` *attribute*,..., *attribute*
`size` *value*

`analyze table` *relation*
`estimate statistics sample` *value* **`percent`**

Tasks

77

Grapefon - schema



Groups – statistics of shared audio recordings

0.1: Create a statistics showing the shared audio content (count only) per group

- Handle groups not having any shared audio content correctly
- Use column alias names
- Format the SQL command

NAME	SHARES_COUNT
PPKE ITK	24
Spirit - Demo	13
teknős6	0
teszt x	0
teknős5	0
[DEL] 3	0
teknős7	0
teszt	429
ía	0
6	0

Groups – statistics of shared audio recordings

0.2. Create a statistics showing the shared audio content (count + average length, rounded to seconds) per group

- Handle groups not having any shared audio content correctly
- Use column alias names
- Format the SQL command

NAME	SHARES_COUNT	SHARES_TOTALLENGTH	SHARES_AVGLENGTH
PPKE ITK	24	7933	331
Spirit - Demo	13	532	41
teknős6	0	(null)	(null)
teszt x	0	(null)	(null)
teknős5	0	(null)	(null)
[DEL] 3	0	(null)	(null)
teknős7	0	(null)	(null)
teszt	429	164076	382
ía	0	(null)	(null)

http://docs.oracle.com/cd/E11882_01/server.112/e26088/functions002.htm#SQLRF51178

Oracle® Database SQL Language Reference 11*g* Release 2 (11.2) Single-Row Functions

Groups, statistics of shared audio recordings and members

0.3. Extend the previous query with an additional column showing the number of the members in the group!

NAME	SHARES_COUNT	SHARES_TOTALLENGTH	MEMBERS_COUNT
teknős6	0	(null)	0
PPKE ITK Énekkar	1	130	5
Emmánuel	56	35442	18
ía	0	(null)	0
[DEL] teknős8	0	(null)	0
[DEL] 3	0	(null)	0
5	0	(null)	0
teknős4	0	(null)	0
Boardport	0	(null)	3

Check the execution plan of the query (number of members, shared audio).

Oracle Documentation: Oracle 12 g Database SQL Tuning Guide, https://docs.oracle.com/en/database/oracle/oracle-database/12.2/tgsql/reading-execution-plans.html#GUID-5AE1939F-F654-42FF-B0C5-706507CD12A2

Check the execution plan of the following for queries! Can you find any differences? What are the reasons for the differences?

```
SELECT COUNT(*)
FROM audio_large
INNER JOIN historyitems_large
ON audio_large.id = historyitems_large.audio_id;

SELECT COUNT(historyitems_large.rating)
FROM audio_large
INNER JOIN historyitems_large
ON audio_large.id = historyitems_large.audio_id;

SELECT AVG(historyitems_large.rating)
FROM audio_large
INNER JOIN historyitems_large
ON audio_large.id = historyitems_large.audio_id;

SELECT COUNT(historyitems_large.updated_at)
FROM audio_large
INNER JOIN historyitems_large
ON audio_large.id = historyitems_large.audio_id;
```

Check the execution plan of the following queries! (For the 2nd and 3rd queries, please only check the execution plans, without starting the queries.) What are the differences? Why?

```
SELECT COUNT(audio_large.user_id)
FROM audio_large
INNER JOIN historyitems_large
ON audio_large.id = historyitems_large.audio_id;

SELECT COUNT(audio_large.user_id)
FROM audio_large
INNER JOIN historyitems_large
ON audio_large.id BETWEEN historyitems_large.audio_id - 0.5 AND historyitems_large.audio_id + 0.5 ;


SELECT COUNT(audio_large.user_id)
FROM audio_large
INNER JOIN historyitems_large
ON audio_large.id - historyitems_large.audio_id = 0;
```

Check changes in the execution plan as the selectivity of the condition in the WHERE clause changes! (Change the condition on the user_id first!)

```
SELECT *
FROM   audio_large
       inner join historyitems_large
               ON audio_large.id = historyitems_large.audio_id
WHERE  To_char(historyitems_large.started_at, 'yyyymmdd') = '20160302'
       AND audio_large.user_id < 500;
```

Database management II.

Objectrelational Databases
Geographical Databases

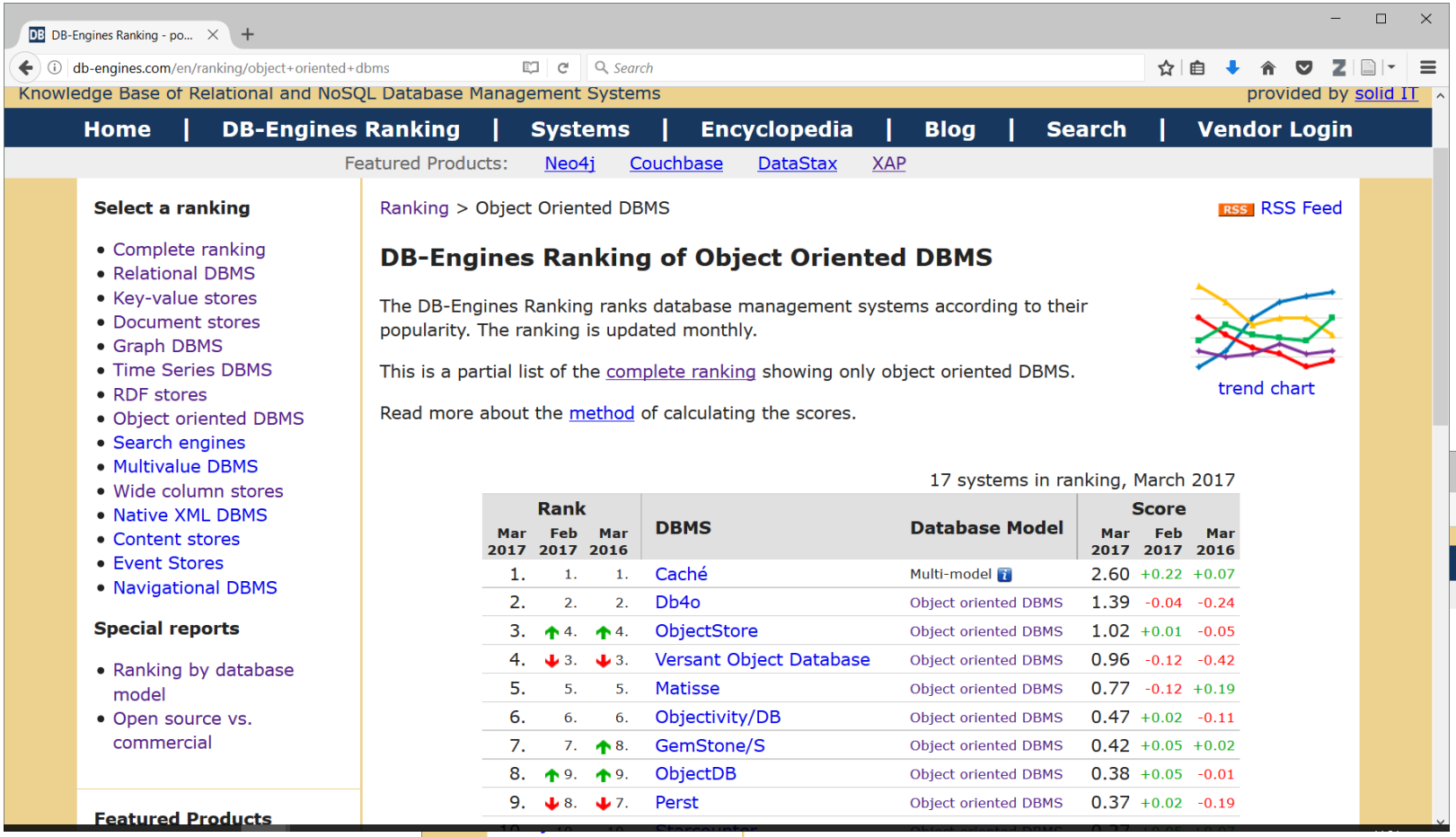
Gergely Lukács
Pázmány Péter Catholic University
Faculty of Information Technology
Budapest, Hungary
lukacs@itk.ppke.hu

Object-Relational Databases

Object-oriented databases („object databases”)

- Object-oriented concepts
  - encapsulation, inheritance, …
- Early 1990s: Research on object-oriented databases
- Object-oriented DBMS’s failed because they did not offer the efficiencies of well-entrenched relational DBMS’s.
- Niche application areas only

http://db-engines.com/en/ranking



DB-Engines Ranking - po...

db-engines.com/en/ranking/object+oriented+dbms

Search

Knowledge Base of Relational and NoSQL Database Management Systems

provided by solid IT

Home | DB-Engines Ranking | Systems | Encyclopedia | Blog | Search | Vendor Login

Featured Products: Neo4j Couchbase DataStax XAP

Select a ranking

- Complete ranking
- Relational DBMS
- Key-value stores
- Document stores
- Graph DBMS
- Time Series DBMS
- RDF stores
- Object oriented DBMS
- Search engines
- Multivalue DBMS
- Wide column stores
- Native XML DBMS
- Content stores
- Event Stores
- Navigational DBMS

Special reports

- Ranking by database model
- Open source vs. commercial

Featured Products

Ranking > Object Oriented DBMS

RSS RSS Feed

DB-Engines Ranking of Object Oriented DBMS

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

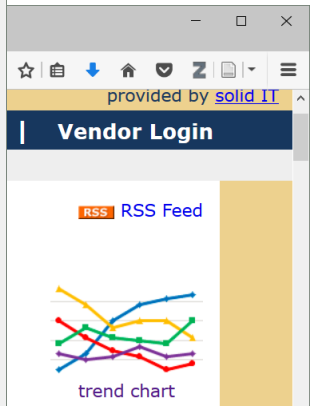
This is a partial list of the complete ranking showing only object oriented DBMS.

Read more about the method of calculating the scores.

trend chart

17 systems in ranking, March 2017

Rank Mar 2017	Rank Feb 2017	Rank Mar 2016	DBMS	Database Model	Score Mar 2017	Score Feb 2017	Score Mar 2016
1.	1.	1.	Caché	Multi-model	2.60	+0.22	+0.07
2.	2.	2.	Db4o	Object oriented DBMS	1.39	-0.04	-0.24
3.	↑ 4.	↑ 4.	ObjectStore	Object oriented DBMS	1.02	+0.01	-0.05
4.	↓ 3.	↓ 3.	Versant Object Database	Object oriented DBMS	0.96	-0.12	-0.42
5.	5.	5.	Matisse	Object oriented DBMS	0.77	-0.12	+0.19
6.	6.	6.	Objectivity/DB	Object oriented DBMS	0.47	+0.02	-0.11
7.	7.	↑ 8.	GemStone/S	Object oriented DBMS	0.42	+0.05	+0.02
8.	↑ 9.	↑ 9.	ObjectDB	Object oriented DBMS	0.38	+0.05	-0.01
9.	↓ 8.	↓ 7.	Perst	Object oriented DBMS	0.37	+0.02	-0.19



provided by solid IT

Vendor Login

RSS RSS Feed

trend chart

- Search engines
- Multivalue DBMS
- Wide column stores
- Native XML DBMS
- Content stores
- Event Stores
- Navigational DBMS

Special reports

- Ranking by database model
- Open source vs. commercial

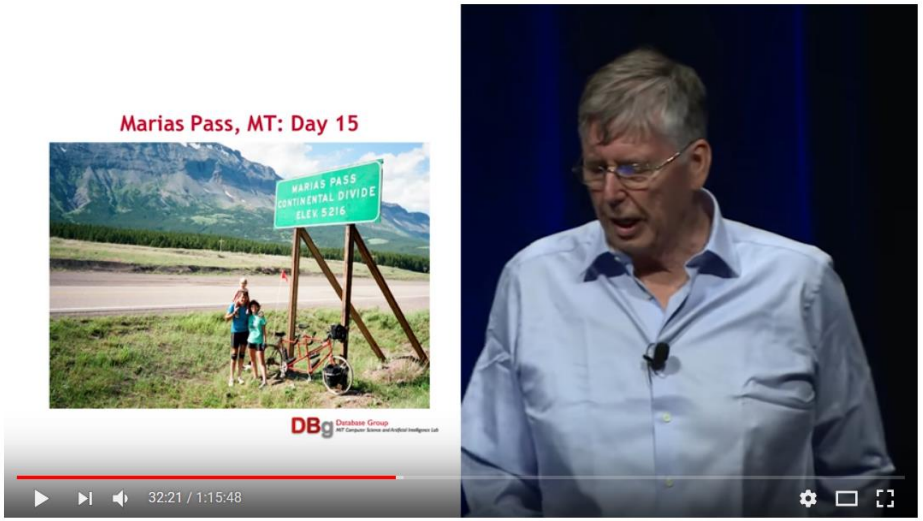
Featured Products

322 systems in ranking, March 2017

Rank Mar 2017	Rank Feb 2017	Rank Mar 2016	DBMS	Database Model	Score Mar 2017	Score Feb 2017	Score Mar 2016
1.	1.	1.	Oracle	Relational DBMS	1399.50	-4.33	-72.51
2.	2.	2.	MySQL	Relational DBMS	1376.07	-4.23	+28.36
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1207.49	+4.04	+71.00
4.	4.	↑ 5.	PostgreSQL	Relational DBMS	357.64	+3.96	+58.01
5.	5.	↓ 4.	MongoDB	Document store	326.93	-8.57	+21.60
6.	6.	6.	DB2	Relational DBMS	184.91	-2.99	-3.02
7.	↑ 8.	7.	Microsoft Access	Relational DBMS	132.94	-0.45	-2.09
8.	↓ 7.	8.	Cassandra	Wide column store	129.19	-5.19	-1.14
9.	9.	↑ 10.	SQLite	Relational DBMS	116.19	+0.88	+10.42
10.	10.	↓ 9.	Redis	Key-value store	113.01	-1.03	+6.79

Object-relational databases

- Michael Stonebraker: 2014 ACM A.M. Turing Lecture, June 13 2015
  https://www.youtube.com/watch?v=BbGeKi6T6QI&t=2867s



- https://amturing.acm.org/award_winners/stonebraker_1172121.cfm
  https://en.wikipedia.org/wiki/Turing_Award

Object-relational databases

- Object-oriented extensions to relational DBMS's
  - advantages of OO, yet retain the relation as the fundamental abstraction.
  - Object-oriented models support interesting data types --- not just flat files.
    - ***Maps, XML, multimedia, etc.***
  - The relational model supports very-high-level queries.

SQL-99 and Oracle Features

- SQL-99 includes many of the object-relational features.
- However, different DBMS's use different approaches.

Oracle: User Defined Types

- A user-defined type, or UDT, is essentially a class definition, with a structure and methods.
- Two uses:
  1. As the type of an attribute of a relation.

```
SQL> CREATE TABLE states (
  2      state          VARCHAR2(30),
  3      totpop         NUMBER(9),
  4      geom           SDO_GEOMETRY);
```

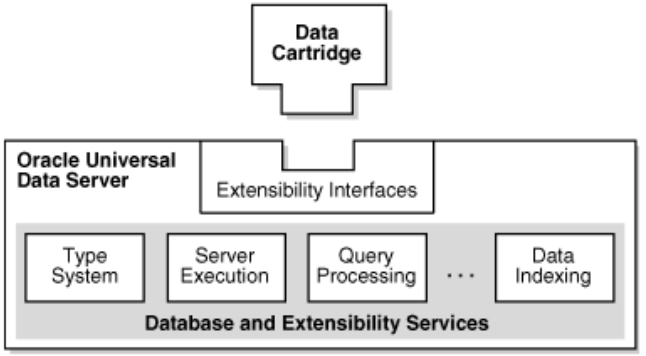
  2. As a rowtype, that is, the type of a relation.

```
create type emp_t as object (empno number, ename varchar2(10), ...);
create table emp of emp_t;
```

Oracle Types, Data Cartride Developers Guide

- In addition to the efficient and secure management of data ordered under the relational model, Oracle provides support for data organized under the object model.

1. Object types, large objects (LOBs),
2. external procedures,
3. extensible indexing
4. query optimization



can be used to build powerful, reusable server-based components called data cartridges.

Objectrelational DBMS (!)

- Extensible data types
- Methods/Operations
- Extensible indexing
  - Cost-based query optimization

How widely used are Oracle objects? (!)

- http://stackoverflow.com/questions/5767200/how-widely-used-are-oracle-objects

- some standard Oracle functionality uses Types, for instance XMLDB and Spatial (which includes declaring columns of Nested Table data types)

- What is **not** commonly done, except it would sadly appear in some college courses, is to use object-based tables instead of regular relational tables to hold regular data like employees and departments
  - **(->object-relational mapper, Hibernate**)
- While these may be nice simple examples to teach the concepts, I fear they may lead to a new generation of database developers who think this approach is suitable, more modern and therefore better than "old-fashioned" relational tables. It emphatically is not.

- Object relational technology adds a huge amount of complexity

Oracle XML DB

XML

Oracle XML DB and XML Developer's Kit enable you to develop high performance applications that process XML content and manage XML stored in the database. XDK and XML APIs allow you to generate and store XML data in the database or in documents outside the database.

- XML DB Developer's Guide
- XML Developer's Kit Programmer's Guide
- XML Java API Reference (Javadoc)
- XML C API Reference
- XML C++ API Reference

Oracle Spatial, …

Oracle Spatial and Location Information

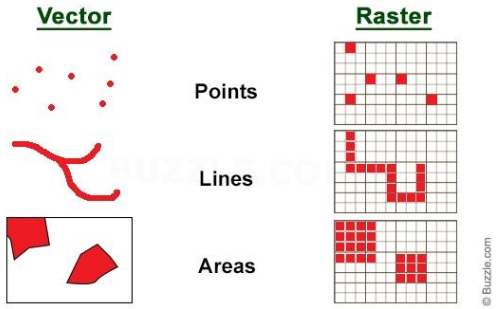
Use features described in these manuals to implement applications that manage data with spatial organization.

Spatial Developer's Guide

Spatial GeoRaster Developer's Guide

Spatial Topology and Network Data Models Developer's Guide

Spatial Java API Reference (Javadoc)



- Oracle Multimedia
- Oracle Text

Table 1-1 Data Cartridge Domains; Content and Scope

Content	Scope: Cross-Industry Uses	Scope: Industry-Specific Extensions
Scalar Data	Statistical conversion	Financial and Petroleum
Multimedia and Complex Unstructured Data	Text	Image
Audio/Video	Spatial	Legal
Medical	Broadcasting	Utilities

PostgreSQL vs Oracle

- Oracle
  - Define type, create table using type
  - Methods
  - No inheritance between tables
  - No multiple parents
  - Smart column types
- PostgreSQL
  - Table inheritance
  - …

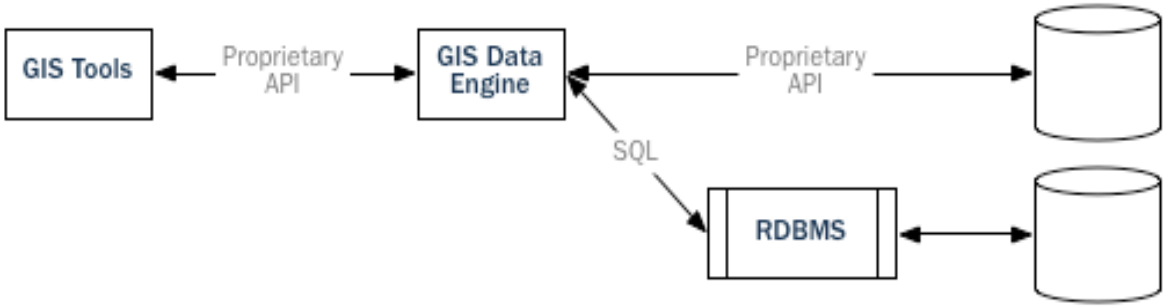
Spatial Databases

Evolution of GIS Architectures

First-Generation GIS:

Geometry, indexes, and attributes stored in file system

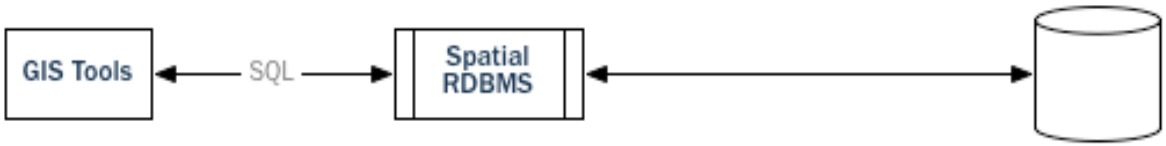
Second-Generation GIS:



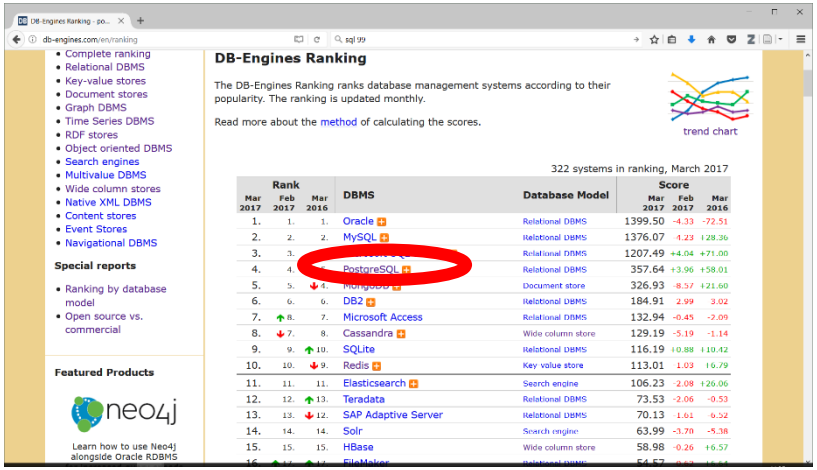
Geometry, indexes, and attributes stored in file system (may also be stored in BLOBs in RDBMS)

Attributes stored in RDBMS tables

Third-Generation GIS:



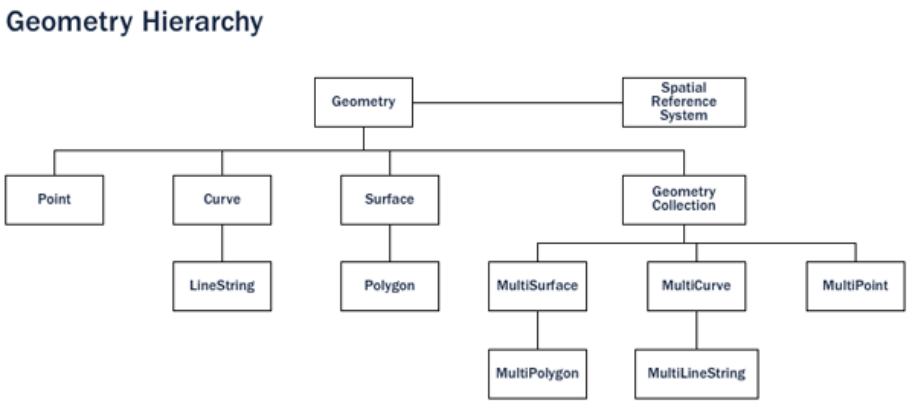
Geometry stored in ADTs in RDBMS tables with related business data

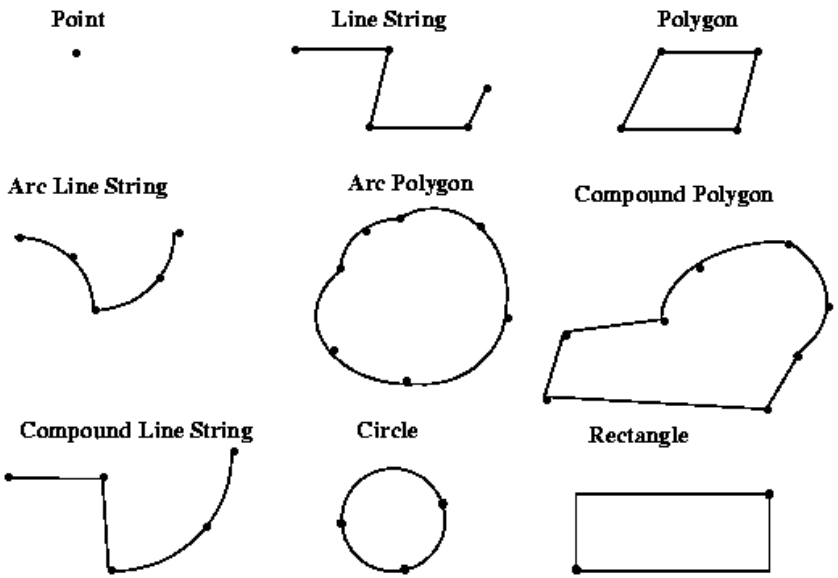


Because the development path for adding types to PostgreSQL was so straightforward, it made sense to start there. When MySQL released basic spatial types in version 4.1, the PostGIS team took a look at their code, and the exercise reinforced the original decision to use PostgreSQL. Because MySQL spatial objects had to be hacked on top of the string type as a special case, the MySQL code was spread over the entire code base. Development of PostGIS 0.1 took under a month. Doing a “MyGIS” 0.1 would have taken a lot longer, and as such, might never have seen the light of day.

PostGIS geographic data types

```
CREATE TABLE testgeog (
 gid serial PRIMARY KEY,
 the_geog geometry( point, 4326 )
);
```





Spatial Reference System, SRID

- World Geodetic System
  - WGS 1984
  - SRID 4326
  - Google, KML

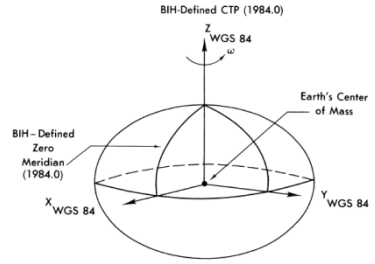
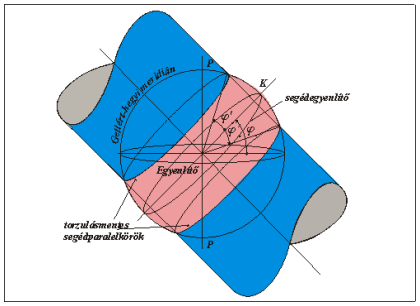
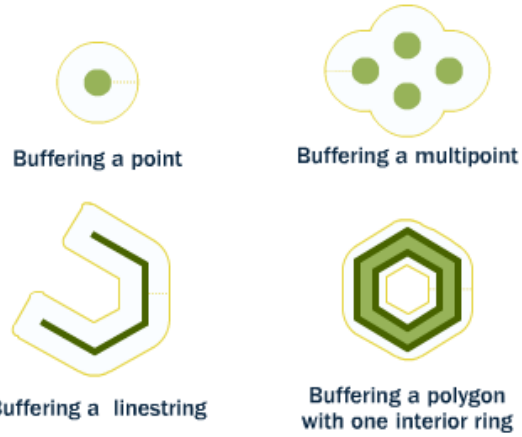


Figure 1.1. WGS 84 Reference Frame

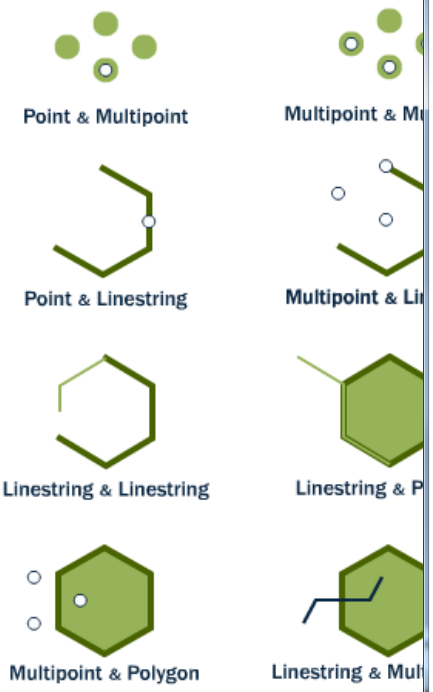


- Hungary: Egységes Országos Vetület
  - SRID 23700
  - https://hu.wikipedia.org/wiki/Egys%C3%A9ges_orsz%C3%A1gos_vet%C3%BClet

ST_Buffer



Intersects



Firefox

Chapter 8. PostGIS Reference

postgis.net/docs/referen

Google

Chapter 8. PostGIS Reference

Prev Next

Chapter 8. PostGIS Reference

Table of Contents

zotero

Firefox

Chapter 8. PostGIS Reference

postgis.net/docs/reference.html#Operators

8.7. Operators

&& — Returns TRUE if A's 2D bounding box intersects B's 2D bounding
&&& — Returns TRUE if A's 3D bounding box intersects B's 3D bounding
&< — Returns TRUE if A's bounding box overlaps or is to the left of B's
&<| — Returns TRUE if A's bounding box overlaps or is below B's.
&> — Returns TRUE if A' bounding box overlaps or is to the right of B's
<< — Returns TRUE if A's bounding box is strictly to the left of B's.
<<| — Returns TRUE if A's bounding box is strictly below B's.
= — Returns TRUE if A's bounding box is the same as B's. Uses double
>> — Returns TRUE if A's bounding box is strictly to the right of B's.
@ — Returns TRUE if A's bounding box is contained by B's.
|&> — Returns TRUE if A's bounding box overlaps or is above B's.
|>> — Returns TRUE if A's bounding box is strictly above B's.
~ — Returns TRUE if A's bounding box contains B's.
~= — Returns TRUE if A's bounding box is the same as B's.
<-> — Returns the distance between two points. For point / point che
types the distance between the floating point bounding box centroids
<#> — Returns the distance between bounding box of 2 geometries. F
geometries are double precision). Useful for doing distance ordering ar

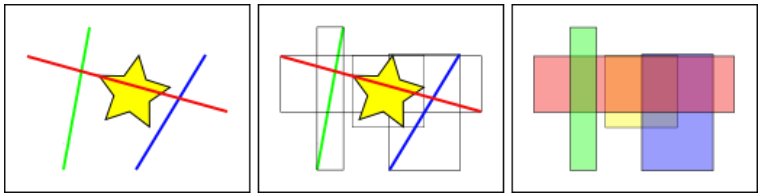
8.8. Spatial Relationships and Measur

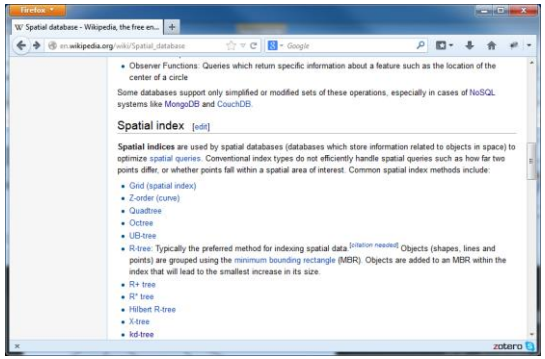
ST_3DClosestPoint — Returns the 3-dimensional point on g1 that is cl
ST_3DDistance — For geometry type Returns the 3-dimensional cartes
ST_3DDWithin — For 3d (z) geometry type Returns true if two geomet
ST_3DDFullyWithin — Returns true if all of the 3D geometries are with
ST_3DIntersects — Returns TRUE if the Geometries "spatially intersect
ST_3DLongestLine — Returns the 3-dimensional longest line between
ST_3DMaxDistance — For geometry type Returns the 3-dimensional c
ST_3DShortestLine — Returns the 3-dimensional shortest line betwee
ST_Area — Returns the area of the surface if it is a polygon or multi-polygon. For "geometry" type area is in SRID units. For "geography" area is in square meters.
ST_Azimuth — Returns the north-based azimuth as the angle in radians measured clockwise from the vertical on pointA to pointB.
ST_Centroid — Returns the geometric center of a geometry.
ST_ClosestPoint — Returns the 2-dimensional point on g1 that is closest to g2. This is the first point of the shortest line.
ST_Contains — Returns true if and only if no points of B lie in the exterior of A, and at least one point of the interior of B lies in the interior of A.
ST_ContainsProperly — Returns true if B intersects the interior of A but not the boundary (or exterior). A does not contain properly itself, but does contain itself.
ST_Covers — Returns 1 (TRUE) if no point in Geometry B is outside Geometry A
ST_CoveredBy — Returns 1 (TRUE) if no point in Geometry/Geography A is outside Geometry/Geography B
ST_Crosses — Returns TRUE if the supplied geometries have some, but not all, interior points in common.
ST_LineCrossingDirection — Given 2 linestrings, returns a number between -3 and 3 denoting what kind of crossing behavior. 0 is no crossing.
ST_Disjoint — Returns TRUE if the Geometries do not "spatially intersect" - if they do not share any space together.
ST_Distance — For geometry type Returns the 2-dimensional cartesian minimum distance (based on spatial ref) between two geometries in projected units. For geography type defaults to return spheroidal minimum distance between two geographies in meters.
ST_HausdorffDistance — Returns the Hausdorff distance between two geometries. Basically a measure of how similar or dissimilar 2 geometries are. Units are in the units of the spatial reference system of the geometries.
ST_MaxDistance — Returns the 2-dimensional largest distance between two geometries in projected units.
ST_Distance_Sphere — Returns minimum distance in meters between two lon/lat geometries. Uses a spherical earth and radius of 6370986 meters. Faster than ST_Distance_Spheroid ST_Distance_Spheroid, but less accurate. PostGIS versions prior to 1.5 only implemented for points.
ST_Distance_Spheroid — Returns the minimum distance between two lon/lat geometries given a particular spheroid. PostGIS versions prior to 1.5 only support points.
ST_DFullyWithin — Returns true if all of the geometries are within the specified distance of one another

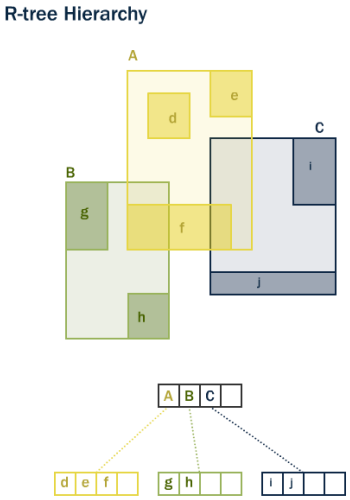
zotero

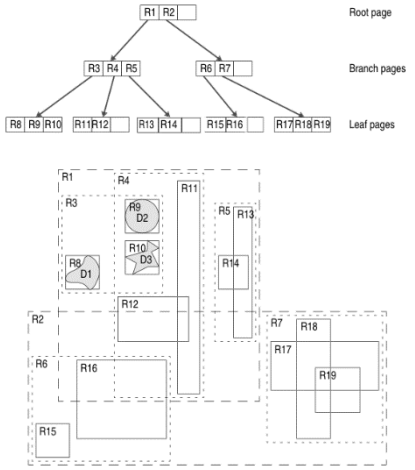
Geographic index (multidimensional index)

- Bounding box
- Tree
  - R-Tree

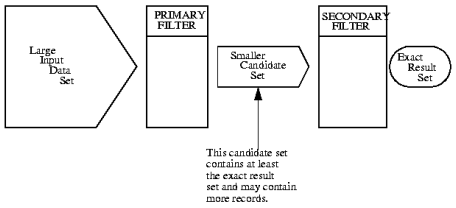


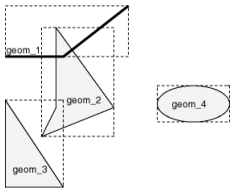


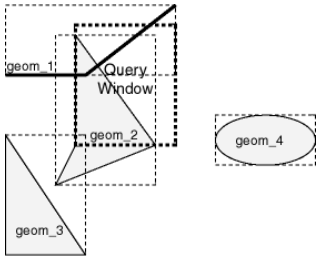




- Query processing
  - Multiple filter sets

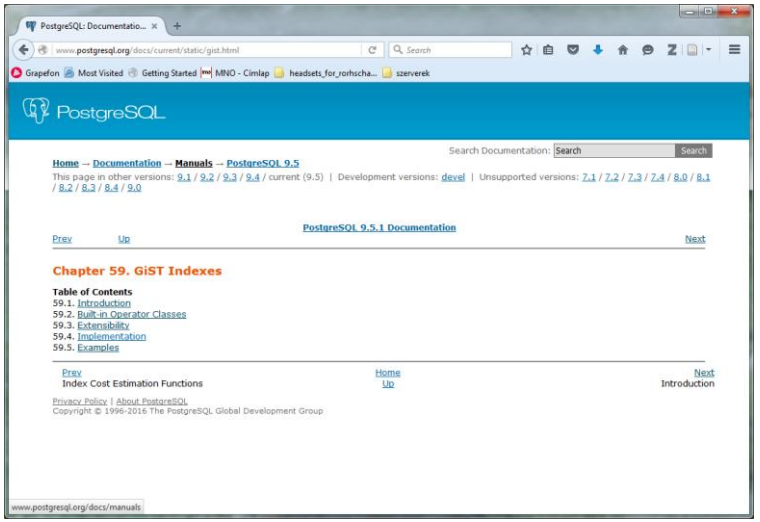






PostgreSQL – Generalized Search Tree

- balanced, tree-structured access method
- template for implementing indexes: B-tree, R-tree,…
- development of custom data types with the appropriate access methods, by an expert in the domain of the data type, rather than a database expert



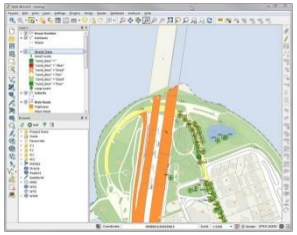
Spatial Database Offerings

- ESRI ArcSDE (on top of several different DBs)
- Oracle Spatial
- IBM DB2 Spatial Extender
- Informix Spatial DataBlade
- MS SQL Server (with ESRI SDE)
- Geomedia on MS Access
- PostGIS / PostgreSQL

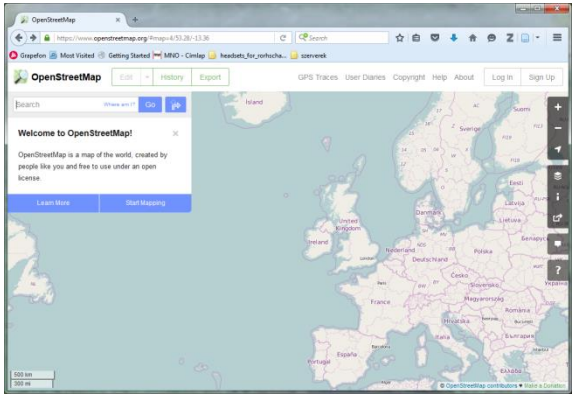
Practical hints

- PostgreSQL + PostGIS (+ pgRouting)

- Visualization: QGIS - Desktop GIS vagy Google



- Data: OpenStreetMap



- Oracle: Check the definition of SYS.XMLTYPE, allowing to manage XML data.

- Use PostgreSQL (9.1) and PostGIS (1.5) for the following tasks!
- Client: pgadmin (locally installed)
- Server connection
  - Host: csquared2.itk.ppke.hu
  - Port: 5432
  - Username: csquared2_db2
  - Maintenance db: csquared2

- It is recommended to use http://sqlformat.darold.net/ for formatting your SQL codes!

- Select a point in a Hungarian city and note its cooordinate values, using Google Maps.

- Check the location 47.501947, 19.034393 on Google Maps!

- Create a point object with SRID set to 4326, using the coordinates 47.501947, 19.034393 (create == query it with a SELECT without a FROM clause, PostgreSQL allows this, Oracle needs the DUAL table).
  - ST_Point
  - Coordinate values swapped! SRID: 4326
  - ST_SetSRID

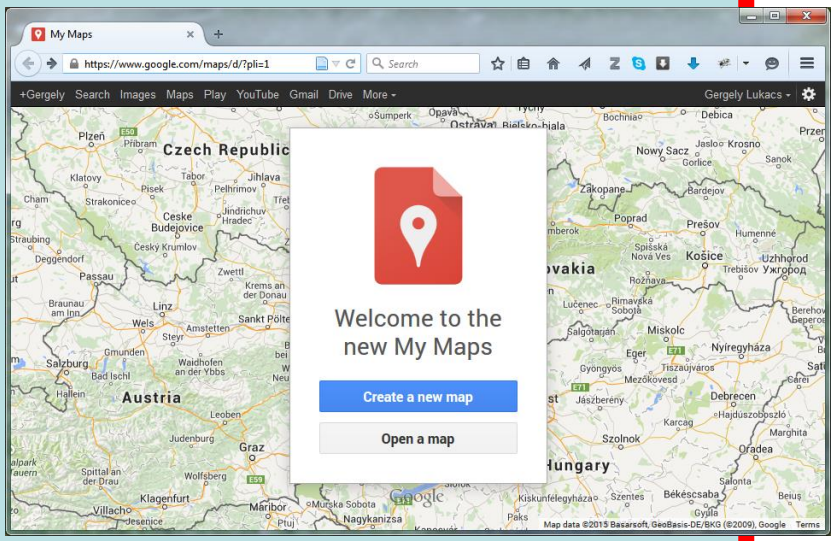
  - CAST the created geometry object to a geography object! http://postgis.net/workshops/postgis-intro/geography.html

  - Check the Well known text (WKT) format of the created geographic object (ST_AsText)

- Calculate (=query) the 2500 meters buffer zone (a geometry!) of the point!
  - (In case of a geogmetry object: ST_Transform;
    Distance can be measured in meters using SRID=23700
    for geography objects not needed!)
  - ST_Buffer

- Create a KML file from the previous buffer zone geometry and visualize it with Google Maps!
  - Creating the KML fragment with PostGIS: **ST_Askml**
  - KML Header, Footer (KML fragment comes to the place of „…”)

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document id="utvonal">
    <name>route</name>
    <Placemark>
        <name>Utvonalnév</name>
        <styleUrl>#LineStyle00</styleUrl>
<MultiGeometry>
…
</MultiGeometry>
<Style id="LineStyle00">
        <LabelStyle>
            <color>00000000</color>
            <scale>0.000000</scale>
        </LabelStyle>
        <LineStyle>
            <color>ff0098e6</color>
            <width>3.000000</width>
        </LineStyle>
    </Style>
  </Placemark>
</Document>
</kml>
```



- Google My Maps, Create a new map, Import
- Check what happens in case of a geometry type without using st_transform! (use 2.5 as radius instead of 2500)

- Query the streets containing in their name „lo” (case-insensitive) and are in the buffer zone calculated previously!
- Aggregate the geometry of the streets and visualize it with Google Maps!
  - Table with streets: hu_2po_4pgr (Open Street Map adat)
  - Attribute
    - name: hu_2po_4pgr.osm_name
    - geometry: hu_2po_4pgr.geom_way
  - ST_Intersects
  - (If data type geometry use ST_Transform)
  - ST_Collect

- Modify the previous query so that the streets (street sections) are ordered accourding to their distance from (47.501947, 19.034393) in increasing order! The result shall also contain the distance in meters!
  - ST_Distance

Pázmány Péter Katolikus Egyetem
Információs Technológiai és Bionikai Kar

JDBC

Készítette: Hegedűs Bettina

Alkalmazás fejlesztési lehetőségek

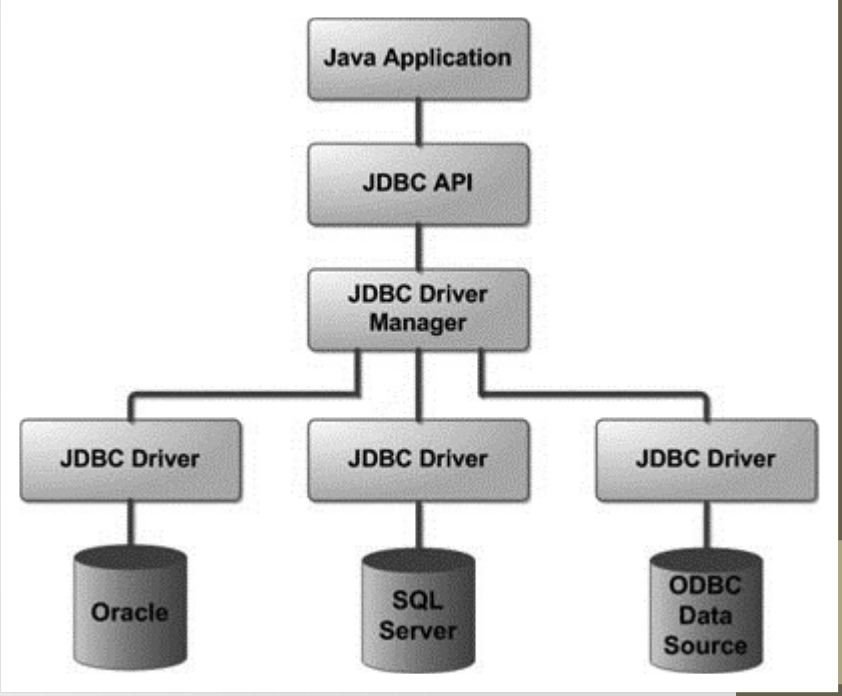
- SQL procedurális kiterjesztései (PL/SQL és társai)
- „Embendded SQL”: az SQL utasításokat beágyazzuk egy host nyelvbe (C,C++ stb.) ezeket fordítás előtt egy előfordító (precompiler) megfelelő függvényhívásokra cseréli.
- Speciális kapcsolódási eszközök, programkönyvtárak alkalmazásával, amelyek segítségével csatlakozni tudunk az adatforráshoz (pl.: ODBC, JDBC)

ODBC és JDBC

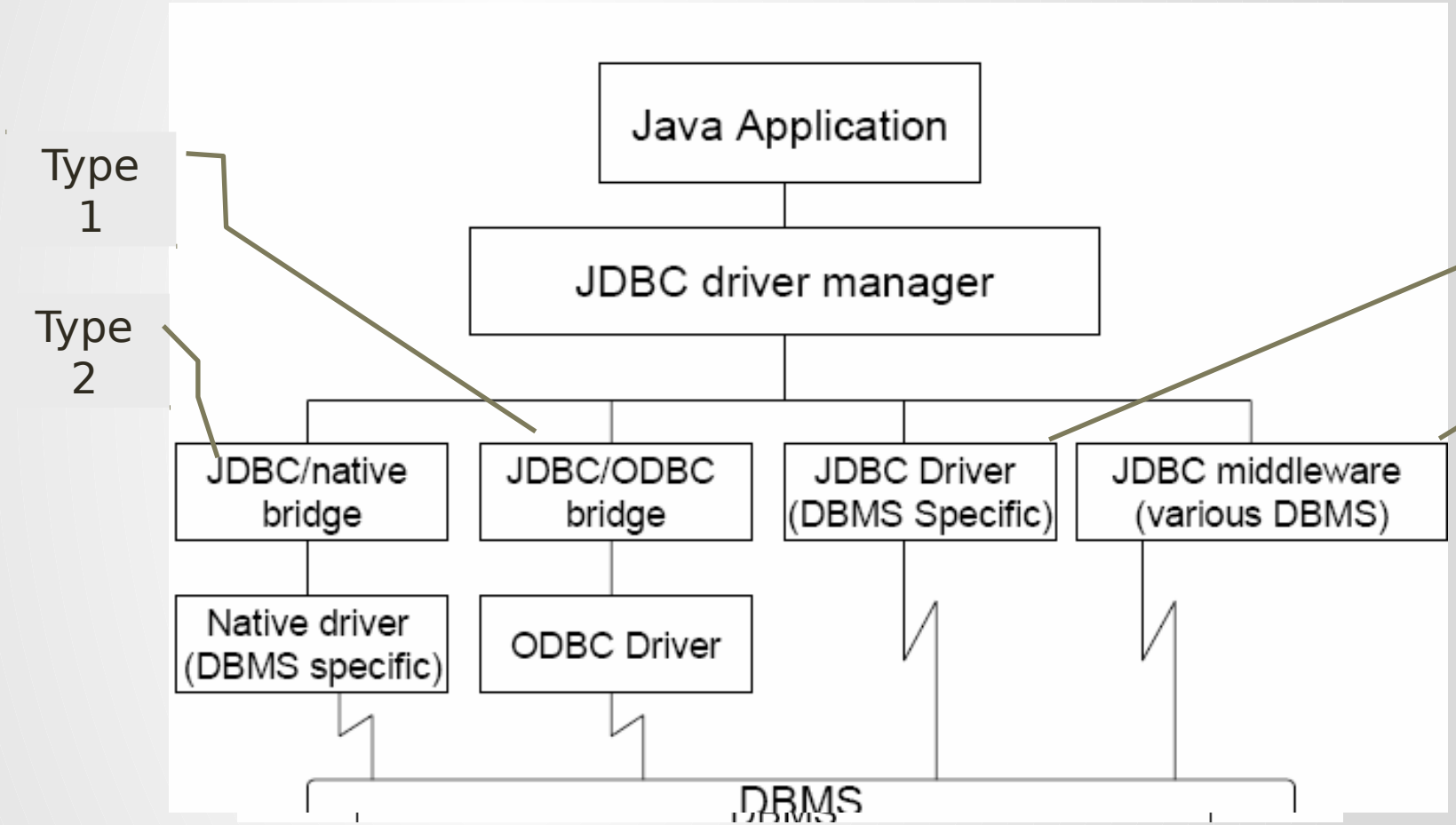
- API (Application Program Interface) mellyel az adatbázis szervert lehet elérni
- A JDBC API szolgáltatásai
  - Kapcsolat létrehozása egy adatforrással(data source)
  - Lekérdező valamint módosító parancsolat lehet küldeni az adatforrásnak
  - Lekérdezések eredményeinek feldolgozása
- Az ODBC (Open Database Connectivity; Microsoft) C, C++, C# és Visual Basic környezetben
- A JDBC (Java Database Connectivity) JAVA-hoz

JDBC felépítése

- Java alkalmazás (kapcsolat nyitás/zárás, SQL kód küldése)
- Driver Manager(ez tölti be az ún. JDBC Drivert)
- Driver(az adatbázis kezelő , adatforrás gyártója adja)
- Data Source (ami végrehajtja a lekérdezést)



JDBC felépítése



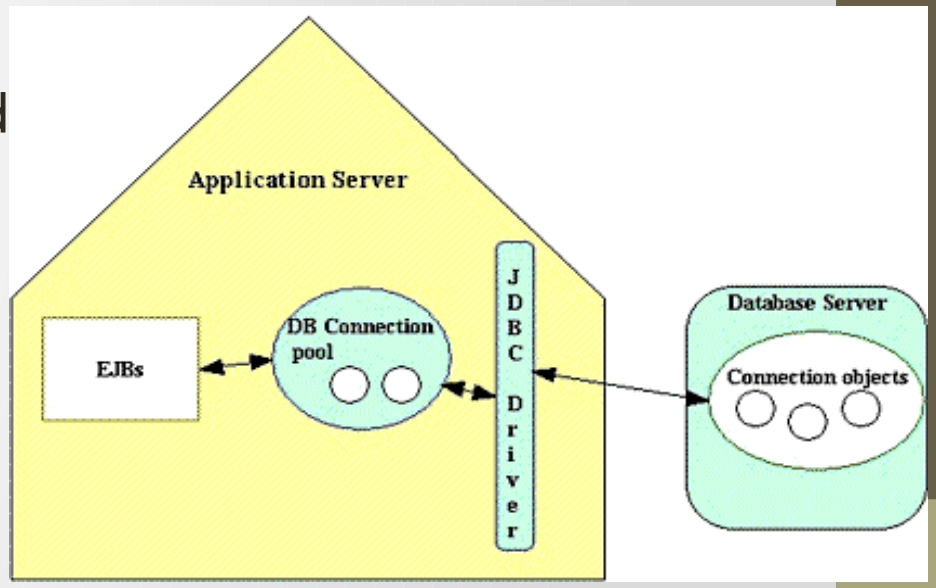
JDBC használati lépései

- Lépések:
  - Betöltjük a Java programunkba a Drivert
  - Csatlakozunk az adatforráshoz
  - Végrehajtjuk az SQL utasítást

  - Részleteiben
    - JVM betölti a Driver interfészt megvalósító osztályt és regisztrálja a DriverManager-nél a DriverManager-től kérünk egy Connection-t
    - A Connectiontól pedig egy Statement-et
    - A Statement-tel hajtjuk végre az SQL utasítást, aminek az eredménye egy ResultSet
    - Ez a ResultSet pedig iterátor segítségével bejárható
    - Végén lezárjuk a kapcsolatot.

JDBC driver betöltés módjai

- DataSource
  - DataSource interfész segítségével az adatforrás használata jobban optimalizálható,
  - adatforrás URL-jét nem szükséges fixen kódolnunk, JNDI-t elég ismerni -> hordozhatóság
- DataManager
  - Mindent kell ismerni a kód
    - host,
    - port,
    - username,
    - password,
    - driver class)



  - PL.:"jdbc:oracle:thin:@oracle.itk.ppke.hu:1521:gyak"

DataSource

```
import java.sql.*;
Import oracle.jdbc.pool.*;

public class ora {
public static void main(String[] args) throws ClassNotFoundException,
  SQLException
{
String url="jdbc:oracle:thin:@oracle.itk.ppke.hu:1521:pract";
String felh="user";
String pass="pass";
OracleDataSource ods=new OracleDataSource();
ods.setURL(url);
ods.setUser(felh);
ods.setPassword(pass);
Connection conn=ods.getConnection();
}
}
```

DriverManager

```
import java.sql.*;
import oracle.jdbc.pool.*;

public class ora{
public static void main(String[] args)
throws ClassNotFoundException, SQLException
{
String url="jdbc:oracle:thin:@oracle.itk.ppke.hu:1521:gyak";
String usr="user";
String pass=„pass";
String adatbazis="oracle.jdbc.driver.OracleDriver";
Class.forName(adatbazis);
Connection conn=
  DriverManager.getConnection(url,usr,pass);
}
}
```

JDBC URL

host:port:service

String
url="**jdbc:oracle:thin:@oracle.itk.ppke.hu:1521:gyak**

JDBC Driver név

Más adatbáziskezelő, más formátumú a JDBC URL

BC url: vigyázat, Oracle Pluggable Database
://technology.amis.nl/2016/01/10/create-jdbc-data-source-or-jdbc-url-database-connection-to-an-oracle-pluggable-database/

Statement használata

Statement:

- executeQuery: a paraméterében megadott SQL lekérdezést végrehajtja, majd az eredménytáblával (ResultSet) tér vissza.

  ```
  Statement stat1 = myCon.createStatement ();
  Stat1.executeQuery("SELECT * FROM test");
  ```

- executeUpdate: a paraméterében megadott SQL utasítást végrehajtja, majd a módosított sorok számával tér vissza.
- execute: a paraméterében megadott SQL utasítást hajtja végre. Az előző két metódus általánosításának tekinthető. Visszatérési értéke true, ha az utasítás által visszaadott eredmény ResultSet típusú, ekkor az a Statement.getResultSet metódussal kérdezhető le.

PreparedStatement:

Paraméterezhető SQL lekérdezés. Előnyei:

- Cache-elődik
- Lekérdezési terv csak egyszer készül el
- SQL Injection ellen véd

```
PreparedStatement stat2 =
myCon.prepareStatement (
"SELECT beer, price FROM Sells WHERE bar
= ?");
```

CallableStatement

Eljárás- és függvényhíváshoz

```
CallableStatement stat3 =
myCon.prepareCall ("{call JoeMenu(?, ?)}");
```

PreparedStatementek hívása

A JDBC megkülönbözteti a lekérdezéseket a módosításoktól:

- **executeQuery()**
- **executeUpdate()**

Query:

```
stat2.setString(1, "Joe’’s Bar");
ResultSet menu = stat2.executeQuery();
```

Update:

```
String sql="INSERT INTO Dept VALUES(?,?,?,?)";
PreparedStatment pstmt=con.prepareStatement(sql);
pstmt.clearParameters();
pstmt.setInt(1,depno);
pstmt.setString(2,depname);
pstmt.setInt(3, depmngr);
pstmt.setString(4,deploc);
int numRows = pstmt.executeUpdate();
```

ResultSet

- A ResultSet tartalmazza a lekérdezések eredményeit, soronként. Lényegében úgy viselkedik, mint egy kurzor/iterátor.
- **next()** metódus meghívásával tovább lép a következő sorra és igaz értékkel tér vissza
- Ha már nincs több elem a ResultSet-ben, akkor hamis értékkel tér vissza
- A ResultSet-ből a **getInt(i), getString(i)** stb. függvényekkel lehet kiszedni az

  adatokat, ahol az "i" az attribútum neve vagy sorszáma a relációban.
- **next()-**en kívül van még, **previous(), last(), first()** stb.
- Egy utasítás végrehajtása akkor zárul le, ha az összes visszaadott eredménytábla fel lett dolgozva (minden sora kiolvasásra került). Az utasítás végrehajtása manuálisan is lezárható a Statement.close metódussal.

ResultSet alkalmazása

```
ResultSet menu =
  stat1.executeQuery("SELECT beer, price
  FROM Sells WHERE bar = 'Joe''s Bar'
  ");

while (menu.next())
{
  theBeer = menu.getString(1);
  thePrice = menu.getFloat("price");
}
```

Tranzakció kezelés

Immediate az alapértelmezés

Autocommit van alapértelmezésben, amit át lehet

természetesen állítani és használni a commit(), rollback()

metódusokat igény szerint

CLOSE() fontos lezárni a kapcsolatot már csak a tranzakció kezelések miatt is

SQL Injection

1 Query is written in the backend script

```
$query = "SELECT user FROM tbl_user WHERE name = '$name' ";
```

2 Hacker inputs admin' OR 1=1 -- in the form field.
So $name=admin' OR 1=1 --

```
$query = "SELECT user FROM tbl_user WHERE name = 'admin' OR 1=1 --' ";
```

```
$query = "SELECT user FROM tbl_user WHERE name = 'admin' OR 1=1  --' ";
```

3 This becomes a logical expression which returns *True* for the query. Hacker can type any query even delete queries in the input string

TechRecite.com

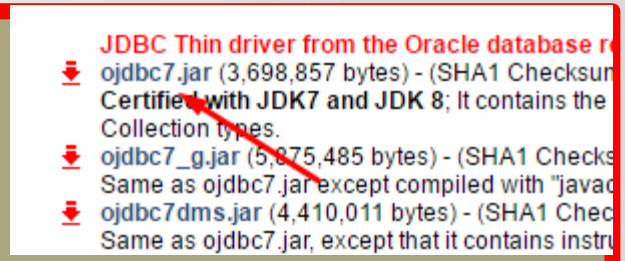
- http://www.programmerinterview.com/index.php/database-sql/sql-injection-example/

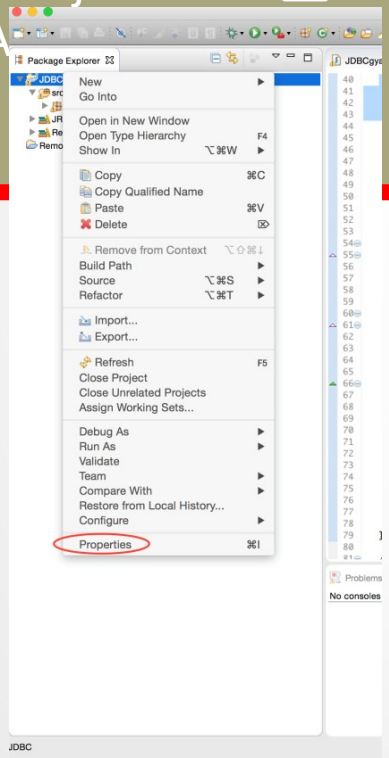
Feladat

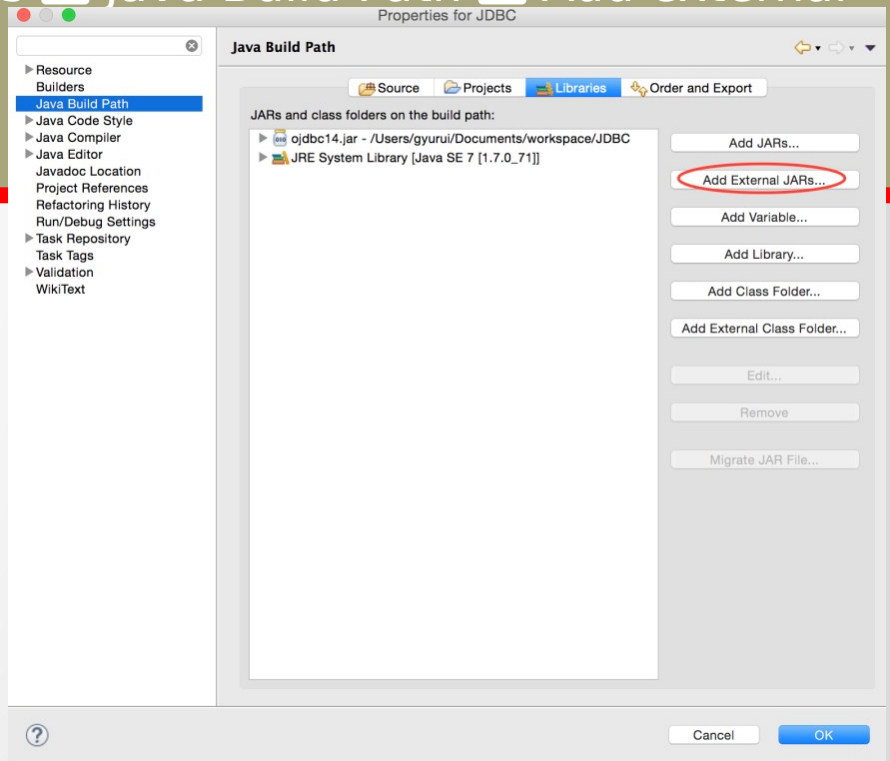
Eclipse megnyitása, új Java projekt

Megfelelő JDBC driver letöltése (JDBC Thin for **12c Release 1**) http://www.oracle.com/technetwork/database/features/jdbc/default-2280470.html

Projekt név ✉ Properties ✉ Java Build Path ✉ Add external JA







Tesztelje a driver működését:
Létesítsen JDBC kapcsolatot az egyetemi Oracle adatbázissal.
Kapcsolati adatok mint az SQL developer-ben (az is egy Java alkalmazás, ami JDBC-n kapcsolódik az Oracle-hez)

JDBC url: vigyázat, Oracle Pluggable Database
https://technology.amis.nl/2016/01/10/create-jdbc-data-source-or-jdbc-url-database-connection-to-an-oracle-pluggable-database/

Hozza létre a következő táblát az adatbázisban. Az id értékeket ne kézzel kelljen megadni.
series tábla:

- series_id NUMBER
- series_name VARCHAR
- description VARCHAR
- next_episode DATE

- Készítsen egy új projektet javaban.
- Készítsen egy *SOROZAT* nyilvántartó alkalmazást a következő specifikáció alapján.
- Lehet gui-s vagy console-os az alkalmazás
- Hint a GUI építéshez:
  - Window Builder , Install new software
  - http://download.eclipse.org/windowbuilder/WB/release/R201406251200/4.4/

A program tudjon hozzáadni a *series* táblához nevet, leírást és következő rész dátumát.

Az alkalmazás tudjon törölni a *series* táblából. (id alapján)

A sorozat neveket egy listába legyen lehetőség lekérdezni.

Egy sorozat név kiválasztásakor jelenjen meg róla a leírás és a következő rész dátuma.

A sorozat dátuma és leírása legyen szerkeszthető.

A feladatokat *prepared statement* segítségével oldja meg.

Készítsen Logot a lekérdezésekhez (írja ki egy fájlba vagy jelenítse meg a console-on).

Pázmány Péter Katolikus Egyetem
Információs Technológiai és Bionikai Kar

Eredetit készítette: Hegedűs Bettina

Object-relational impedance mismatch

- Identity
  - Primary key attribute(s)
- Data type differences
  - Strings
- Manipulative differences
- Transactional differences
  - DBMS: transactions, OO: individual operations
- …

ORM

ORM (Object Relational Mapping)- Objektum relációs leképzés

Az ORM egy programozási technika, ami objektum orientált rendszert hivatott leképezni (nem kompatibilis) rendszerbe (pl: relációs adatbázis).

A cél az, hogy az objektumok tulajdonságait és kapcsolatait megőrizzük, így helyesen vissza tudjuk tölteni azokat az adatbázisba.

Object-Relational Mapping

- Kétirányú leképezés az objektumorientált világ és a relációs modell között
  - One class – one table (a legegyszerűbb esetben)
  - One instance – one row
- Egyszerű példa:

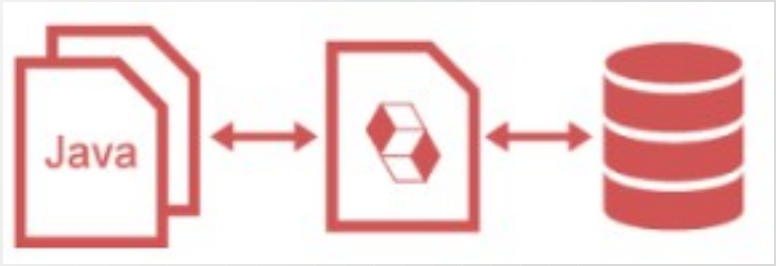
```
Statement cmd = conn.createStatement(
   "SELECT id, first_name, last_name, phone FROM
   persons WHERE id = 10";);
ResultSet res = cmd.executeQuery();
res.next();
Person p = new Person(res.getString("
   first_name"));
```

JDBC helyett Hibernate-tel:

```
Person p = session.get(Person.class, 10);
```

Mi a Hibernate?

- Hibernate egy teljesen Java alapú framework és egy ORM , amely lehetővé teszi az objektumrelációs leképezést:
  POJO-k relációs adatbázishoz való kapcsolását.
- Nagyon hatékony leképzést hoz létre a java objektumok és az adatbázis táblák között
- Hibernate-et használva az alkalmazásfejlesztést nagyon le lehet egyszerűsíteni, fel lehet gyorsítani



Hibernate Architekturális felépítése

Kommunikációs egységek

- *SessionFactory*: Szálbiztos objektum Session-ök létrehozására. Az alkalmazások az ezt az interfészt implementáló osztály egyetlen példányát használják.
- *Session*: Ennek az interfésznek az implementációi biztosítják az entitásokon való műveletek végzését.A kliens és az adatbázis között egy párbeszédet reprezentál. Mikor adatbázis módosításokat akarunk végezni, a Session objektumok létrehoznak egy Transaction típusú objektumot. Ezek az objektumok reprezentálják az adatbázison elvégzett munkát. A Session objektum nem szálbiztos, csak egy kliens használhatja egyszerre.
- *Perzisztens objektumok*: rövid életű objektum, egy Session-höz tartozhatnak csak. Ha a Session bezárul, az objektum elérhetıvé válik az alkalmazás többi rétege számára.

Hibernate felépítése

- 3 rész:
  - Java osztály
  - Relációs adatbázisbeli táblák
  - Leíró (descriptor)
    - Definiálja a konverziós szabályokat
    - 2 fajtája van:
      - XML fájlok (valaminév.hbm.xml kiterjesztéssel)
      - Annotációk használata

Hibernate előnyei

- Objektumokat használ táblák helyett, objektum perzisztencia
- Rövidebb fejlesztési idő
- Könnyen karbantartható
- Cache-elés (lazy vagy eager)
- Version property (optimistic locking)
- Adatbáziskezelőtől független fejlesztés
- Connection pooling

- Persze futásnál valamennyi overhead-et jelent

(Main)

```
SessionFactory sessionFactory;
ServiceRegistry serviceRegistry;

Configuration configuration=new Configuration();
configuration.configure();
serviceRegistry = new StandardServiceRegistryBuilder().applySettings(
configuration.getProperties()).build();
sessionFactory = configuration.buildSessionFactory(serviceRegistry);

User user1=new User();
user1.setUserName(”Lajos");

Session ss= sessionFactory.openSession();
ss.beginTransaction();
ss.save(user1);
ss.save(user2);
ss.getTransaction().commit();
ss.close();
sessionFactory.close();
```

(Konfigurációs fájl)

- **hibernate.cfg.xml**
- **Létre kell hozni a projectben új fájlként**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <property name="connection.url">jdbc:oracle:thin:@oracle.itk.ppke.hu:1521:gyak</property>
        <property name="connection.username">nevem</property>
        <property name="connection.password">jelszvam</property>
        <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>
        <!-- SQL dialect -->
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>
        <!-- Drop and re-create the database schema on startup -->
        <property name="hbm2ddl.auto">create</property>
        <!-- Magic -->
    <property name="hibernate.temp.use_jdbc_metadata_defaults">false</property>
<mapping class="package.UserClass"></mapping>
 </session-factory>

</hibernate-configuration>
```

JDBC URL kicserélendő a JDBC-nél használtra, ami az Oracle Pluggable Database-hez való

(XML)

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="hibernate02XML.DBUser" table="DBUSER">
        <id name="userId" type="int">
            <column name="USER_ID" precision="5" scale="0" />
            <generator class="assigned" />
        </id>
        <property name="username" type="string">
            <column name="USERNAME" length="20" not-null="true" />
        </property>
        <property name="createdBy" type="string">
            <column name="CREATED_BY" length="20" not-null="true" />
        </property>
        <property name="createdDate" type="date">
            <column name="CREATED_DATE" length="7" not-null="true" />
        </property>
    </class>
</hibernate-mapping>
```

(Annotációk)

Az @ anotációkat a Java 5.0-ás verziójánál vezették be. Az annotációk a programkód elemeihez rendelhetők (csomagokhoz, típusokhoz, metódusokhoz, attribútumokhoz, konstruktorokhoz, lokális változókhoz), plusz információt hordoznak a Java fordító ill. speciális eszközök számára.

```
@Entity(name="User_table")
public class User {

  @Id @GeneratedValue  // primary key és generáljuk a keyt
  int userId;

  @Column(name="User_Name") // adatbázisban oszlop neve
  String userName;

  ....stb
}
```

További:

https://docs.jboss.org/hibernate/stable/annotations/reference/en/html/entity.html

(HQL)

Egy egyszerű példa

```
Session ss = sessionFactory.openSession();
        Transaction tx = null;
        try{
          tx = ss.beginTransaction();
          List choclate = ss.createQuery("FROM Chocolates").list();
          for (Iterator iterator1 =
                        choclate.iterator(); iterator1.hasNext();){
            Chocolates csoki = (Chocolates) iterator1.next();
            System.out.print("Neve:  " + csoki.getFlavour());
          }

          tx.commit();
        }catch (HibernateException e) {
          if (tx!=null) tx.rollback();
          e.printStackTrace();
        }finally {
          ss.close();
        }
```

Kapcsolatok

- @OneToOne: 1-1 kapcsolat megvalósítása. Alapértelmezetten kapcsoló mezővel (join column) kerül leképezésre.
- @ManyToOne: N-1 kapcsolat megvalósítása, az N oldali entitásban. Alapértelmezetten kapcsoló mezővel (join column) kerül leképezésre.
- @OneToMany: 1-N kapcsolat megvalósítása, az 1 oldali entitásban. Az adattag típusa a kapcsolódó entitás típusából készített kollekció (List, Set). Alapértelmezetten kapcsoló táblával (join table) kerül leképezésre, de legtöbbször érdemesebb az N oldali entitás táblájába illesztett kapcsoló mezővel megvalósítani.
- @ManyToMany: N-M kapcsolat megvalósítása. Az adattag típusa a kapcsolódó entitás típusából készített kollekció (List, Set). Alapértelmezetten kapcsoló táblával (join table) kerül leképezésre, de sok esetben az N-M kapcsolatoknak saját tulajdonságai vannak, ami új köztes entitás bevezetését igénylik

```
<property name="studentName" type="string" not-null="true" length="100" column="STUDENT_NAME" />
<set name="studentPhoneNumbers" table="STUDENT_PHONE" cascade="all">
    <key column="STUDENT_ID" />
    <many-to-many column="PHONE_ID" unique="true" class="hibernate.Phone" />
</set>
```

```
@OneToMany(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
@JoinTable(name = "beerFac_beer", joinColumns = {
        @JoinColumn(name = "beerFactory_id", nullable = false, updatable = false) }, inverseJoinColumns = {
                @JoinColumn(name = "beer_id", nullable = false, updatable = false) })
```

- Kapcsolat betöltésének két féle módja:
  - Mohó
  - Lusta

(Példa)

```java
@Id @GeneratedValue
@Column(name = "USER_ID", nullable = false)
public int getUserId() {
    return userId;
}

public void setUserId(int userId) {
    this.userId = userId;
}
```

Tábla név

```java
@Entity
@Table(name = "DBUSER")
public class DBUser {

    private int userId;
    private String userName;
    private String createdBy;
    private Date createdDate;
```

Id : Elsődleges kulcs az adatbázisban

Getter-Setter : minden adattaghoz

```java
SessionFactory sessionFactory;
StandardServiceRegistry serviceRegistry;

Configuration configuration=new Configuration();
configuration.configure();
serviceRegistry = new StandardServiceRegistryBuilder().applySettings(
configuration.getProperties()).build();
sessionFactory = configuration.buildSessionFactory(serviceRegistry);

DBUser user = new DBUser();
DBUser user2 = new DBUser();
Session ss = sessionFactory.openSession();
ss.beginTransaction();
user.setUserName("testName");
user.setCreatedBy("system");
user.setCreatedDate(new java.util.Date(2010, 10, 10));
user2.setUserName("testName2");
user2.setCreatedBy("system2");
user2.setCreatedDate(new java.util.Date(2016, 10, 10));
ss.save(user);
ss.save(user2);
ss.getTransaction().commit();
ss.close();
sessionFactory.close();
```

Adattal való feltöltés

Segítség a feladatokhoz

- **http://www.tutorialspoint.com/hibernate/index.htm**
- **http://www.mkyong.com/hibernate/hibernate-many-to-many-relationship-example-annotation/**
- https://hu.wikibooks.org/wiki/Hibernate_annot%C3%A1ci%C3%B3k_-_entit%C3%A1s_asszoci%C3%A1ci%C3%B3k/kapcsolatok

Töltsd le a Hibernate csomagot a következő linkről:
https://sourceforge.net/projects/hibernate/files/hibernate-orm/5.4.8.Final/
Készíts egy új java projectet tetszőlegesen választott néven.

- Add hozzá a hibernate csomagban található required .jar fájlokat a projeckthez
- Add hozzá a korábbi órán letöltött jdbc fájlt a projekthez

- Hozz létre egy **hibernate.cfg.xml** fájlt és állítsátok be az előadáson hozott példa szerint.
- Hozz létre egy **Categories** nevű osztályt aminek legyen egy **categoryId** egy **name** és egy **longname** objektuma
- A **categoryId** legyen elsődleges kulcs és az adatbázis generálja nekünk ne nekünk kelljen megadni.
- A táblában legyen a longname nevű oszlop ami a kódban a description néven szerepeljen
- A táblában legyen a két kedvenc csokid!

- Hozz létre egy egy-sok kapcsolatban álló blogentry és blogentrycomment táblát között Hibernate segítségével
- A blogentry-nek legyenek a következő attribútumai:
  - Id (primary key), name és longname és a blogentrydate
- blogentrycomment -nak legyen:
  - Id-ja, commenttext és commentdate attribútuma

- Töltsd fel néhány próbabejegyzést
- Ellenőrizd sqldeveloperen keresztül hogy minden rendben működött!!
- Az egyetlen kikötés ezen kívül hogy az adatbázisban lévő elemek ne törlődjenek minden futtatás után.

Database management II.

Data warehousing

Gergely Lukács
Pázmány Péter Catholic University
Faculty of Information Technology
Budapest, Hungary
lukacs@itk.ppke.hu

INTRODUCTION

Motivation

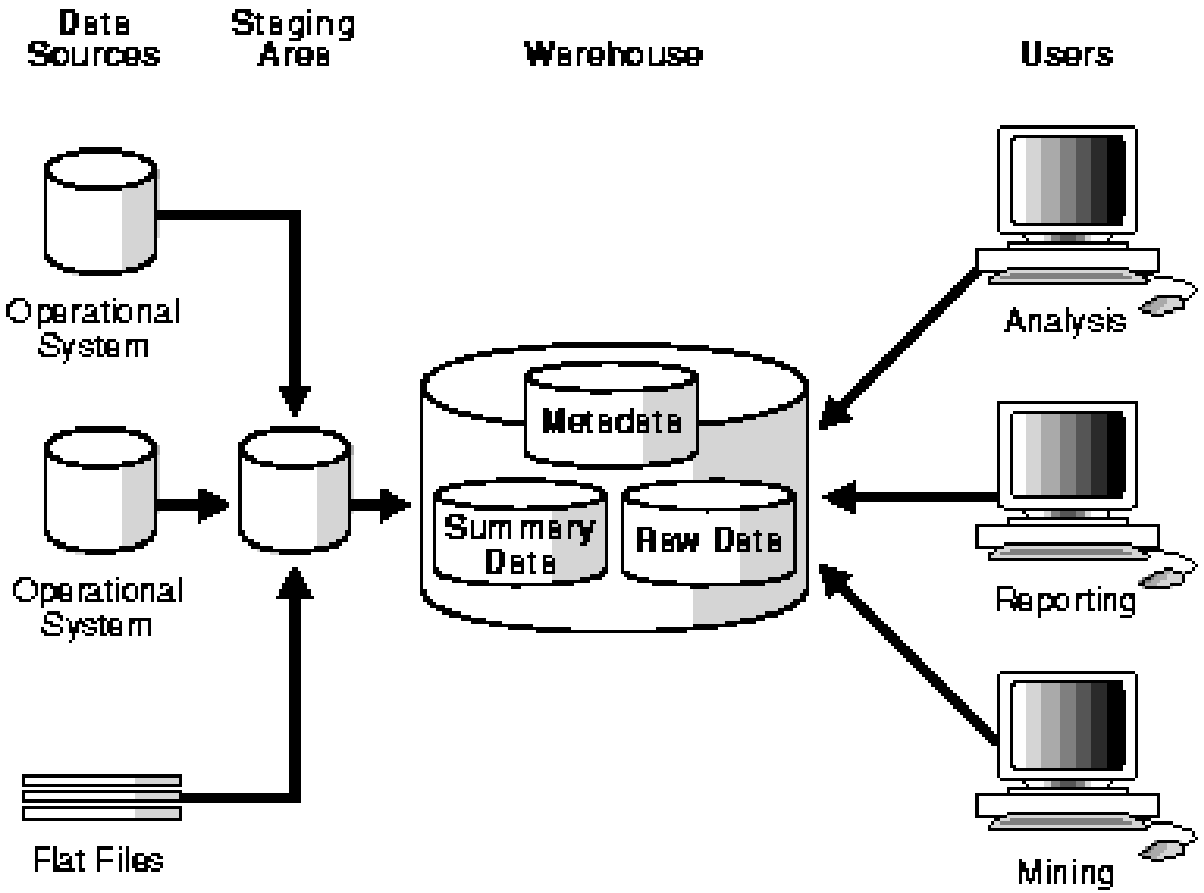
- Up to now:
  - Supporting individual, standard operations (buying, transferring money, lending a book in a libary,…)
  - Standardized business processes
  - Small transactions with data modifications
  - folyamatok
- Now (incl. Tableau)
  - Supporting strategic decisions
  - When should the library be open?
  - Which books shell we buy?
  - Trends
    - How does this change in time, space and over organisational units

Definition

- **A data warehouse is a copy of transaction data specifically structured for query and analysis. (Ralph Kimball)**

Definition - 2 (Gartner Group)

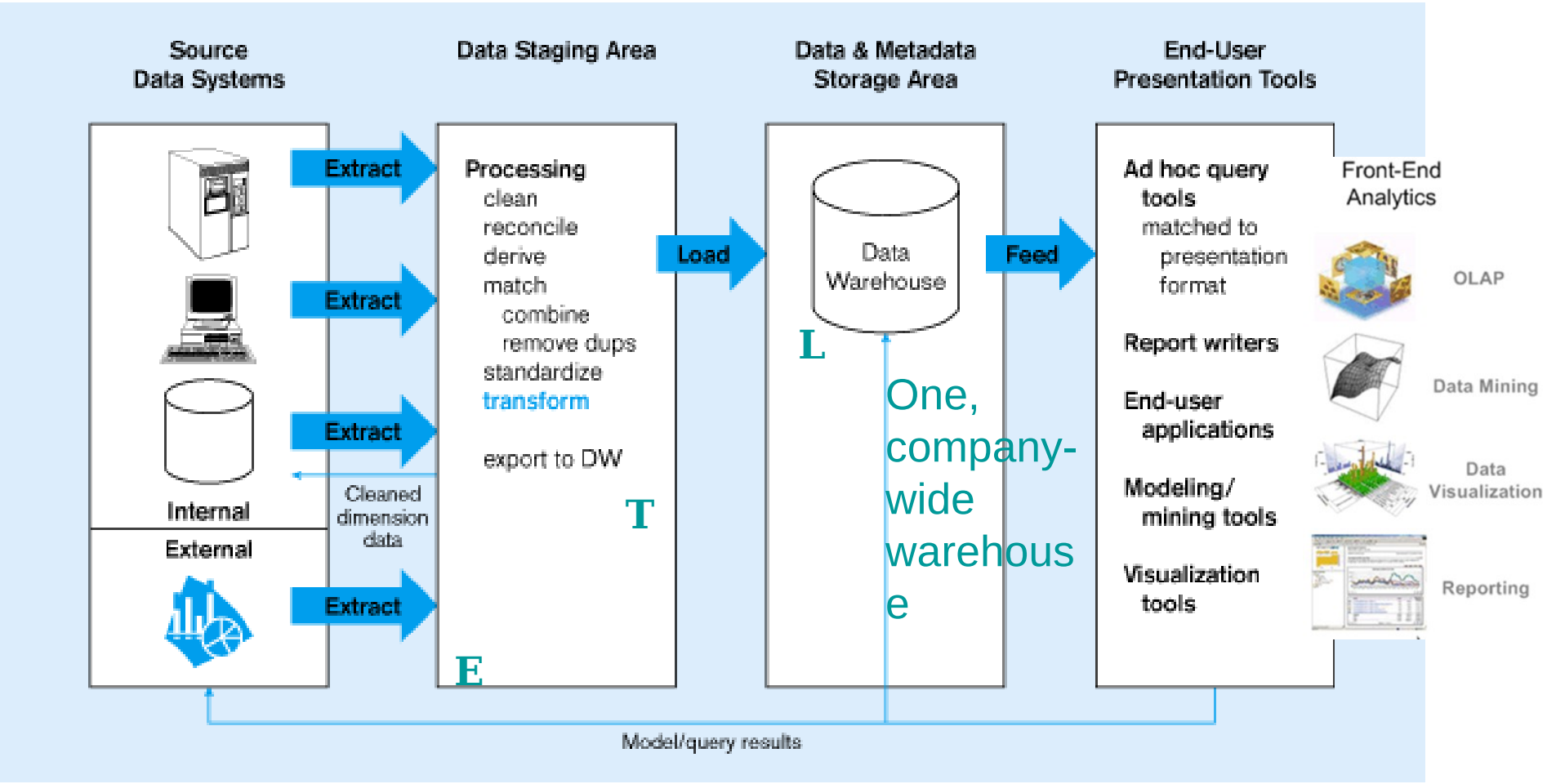
- A **data warehouse** is a storage architecture designed to hold data extracted from transaction systems, operational data stores and external sources. The warehouse then combines that data in an aggregate, summary form suitable for enterprise-wide data analysis and reporting for predefined business needs.


Data Sources
Staging Area
Warehouse
Users
Operational System
Operational System
Flat Files
Metadata
Summary Data
Raw Data
Analysis
Reporting
Mining

Definition -2.2 (Gartner Group)

- The five components of a data warehouse are
  - production data sources,
  - data extraction and conversion,
  - the data warehouse database management system,
  - data warehouse administration and
  - business intelligence (BI) tools.

Figure 11-2: Generic two-level architecture



Periodic extraction ➔ data is not completely current in warehouse

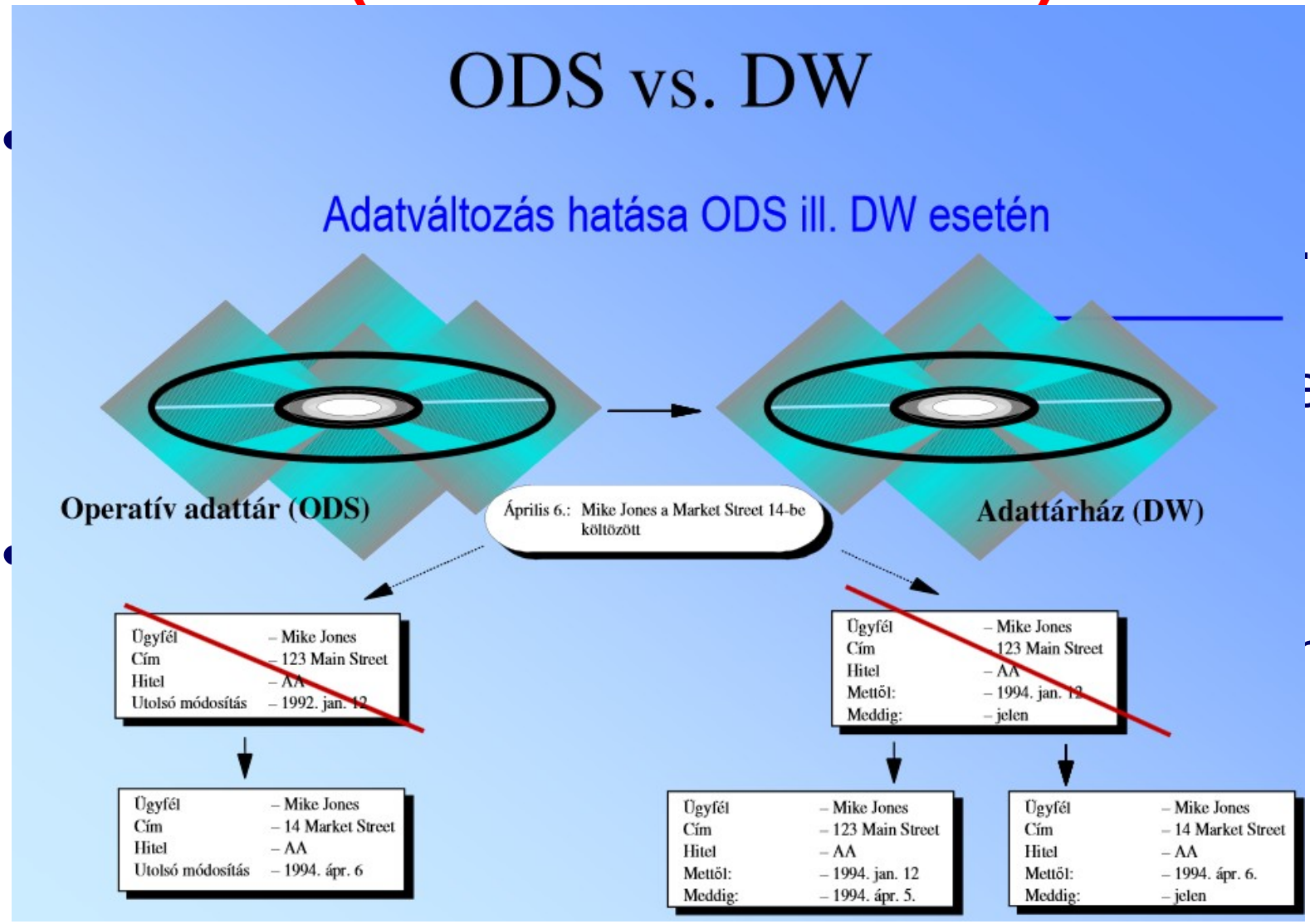
	Operatív adatkezelés	Döntéstámogató adatkezelés
Funkció	Adatfeldolgozás	Döntéstámogatás
Szállítandó	Üzleti műveletek támogatása	Információ
Szervezés	Folyamatorientált	Témaorientált
	Stabil, jól ismert folyamatokhoz	Ismeretlen, feltáró folyamatokhoz
Séma	Normalizált, sok tábla, kevés oszloppal	Kevesebb tábla, leíró oszlopok, redundancia
Lekérdezési profil	Ismert, ismétlődő lekérdezések	Aggregálások, sokféle lekérdezés
Érintett rekordszám	Kevés	Igen nagy
Válaszidő	Nagyon gyors	Lassabb
Frissítés	Folyamatos frissítés, sok tranzakció	Periodikus frissítás, bulk loading
Felhasználók száma	Nagy, egyidejűleg is	Jóval kevesebb
Számításigény	Stabil	Nagyon változó
Rendelkezésre állás	Kritikus, közel 100%	Nem kritikus

(Definition - 3)

(Bill Inmon)

- **Subject-Oriented**:
  - A data warehouse can be used to analyze a particular subject area. For example, "sales" can be a particular subject.
- **Integrated**:
  - A data warehouse integrates data from multiple data sources. For example, source A and source B may have different ways of identifying a product, but in a data warehouse, there will be only a single way of identifying a product.

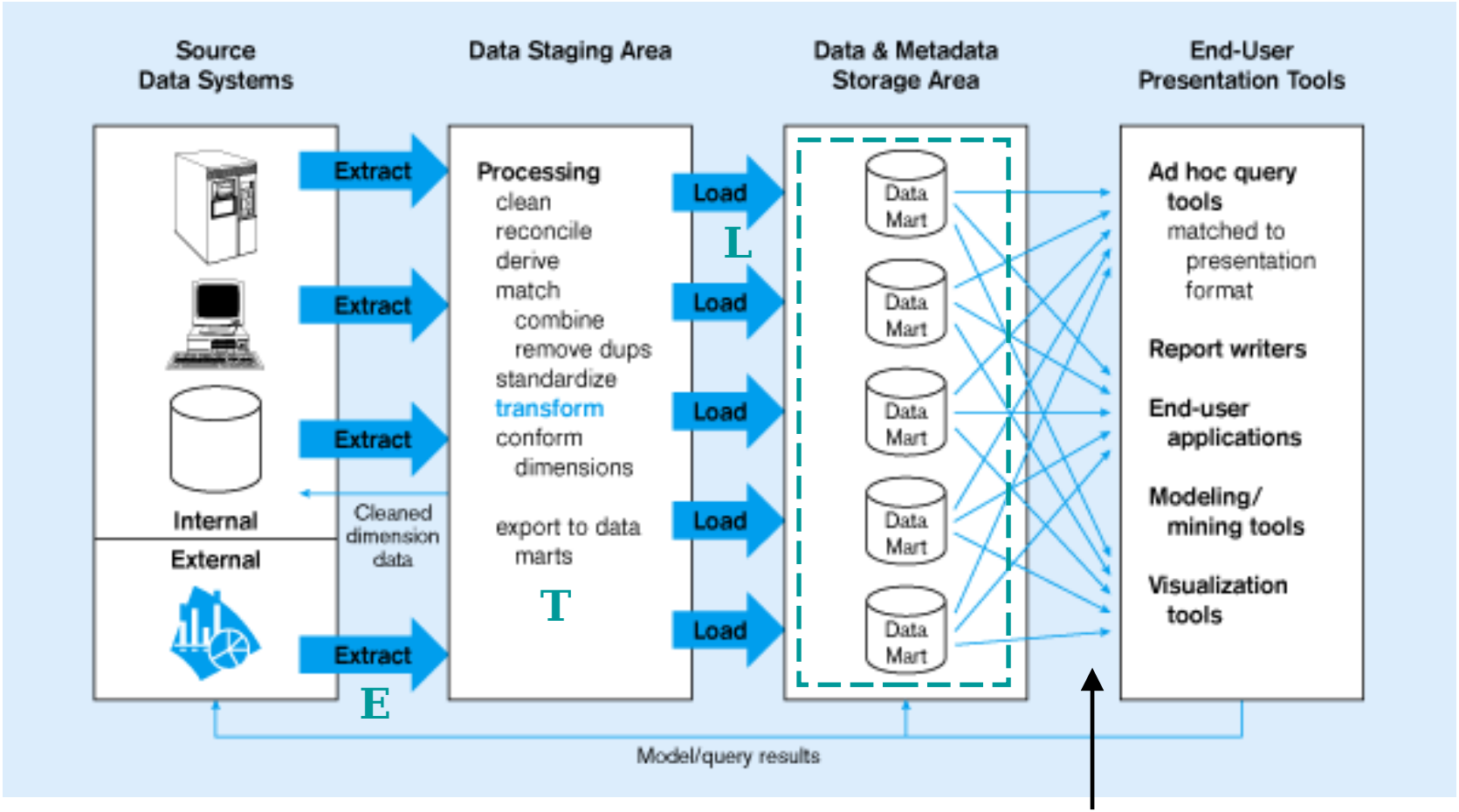
(Definition - 3.2)



(DATA MARTS)

Figure 11-3: Independent Data Mart

Data marts:
Mini-warehouses, limited in scope



Separate ETL for each ***independent*** data mart

Data access complexity due to ***multiple*** data marts

Table 11-2: Data Warehouse vs. Data Mart

Table 11-2 Data Warehouse Versus Data Mart

Data Warehouse	*Data Mart*
Scope	*Scope*
• Application independent	• Specific DSS application
• Centralized, possibly enterprise-wide	• Decentralized by user area
• Planned	• Organic, possibly not planned
Data	*Data*
• Historical, detailed, and summarized	• Some history, detailed, and summarized
• Lightly denormalized	• Highly denormalized
Subjects	*Subjects*
• Multiple subjects	• One central subject of concern to users
Sources	*Sources*
• Many internal and external sources	• Few internal and external sources
Other Characteristics	*Other Characteristics*
• Flexible	• Restrictive
• Data-oriented	• Project-oriented
• Long life	• Short life
• Large	• Start small, becomes large
• Single complex structure	• Multi, semi-complex structures, together complex

Adapted from Strange (1997)

Source: adapted from Strange (1997).

ETL PROCESSES, TOOLS

Data Reconciliation

- Typical operational data is:
  - Transient – not historical
  - Restricted in scope – not comprehensive
  - Sometimes poor quality – inconsistencies and errors
- After ETL, data should be:
  - Detailed – not summarized yet
  - Historical – periodic
  - Comprehensive – enterprise-wide perspective
  - Quality controlled – accurate with full integrity

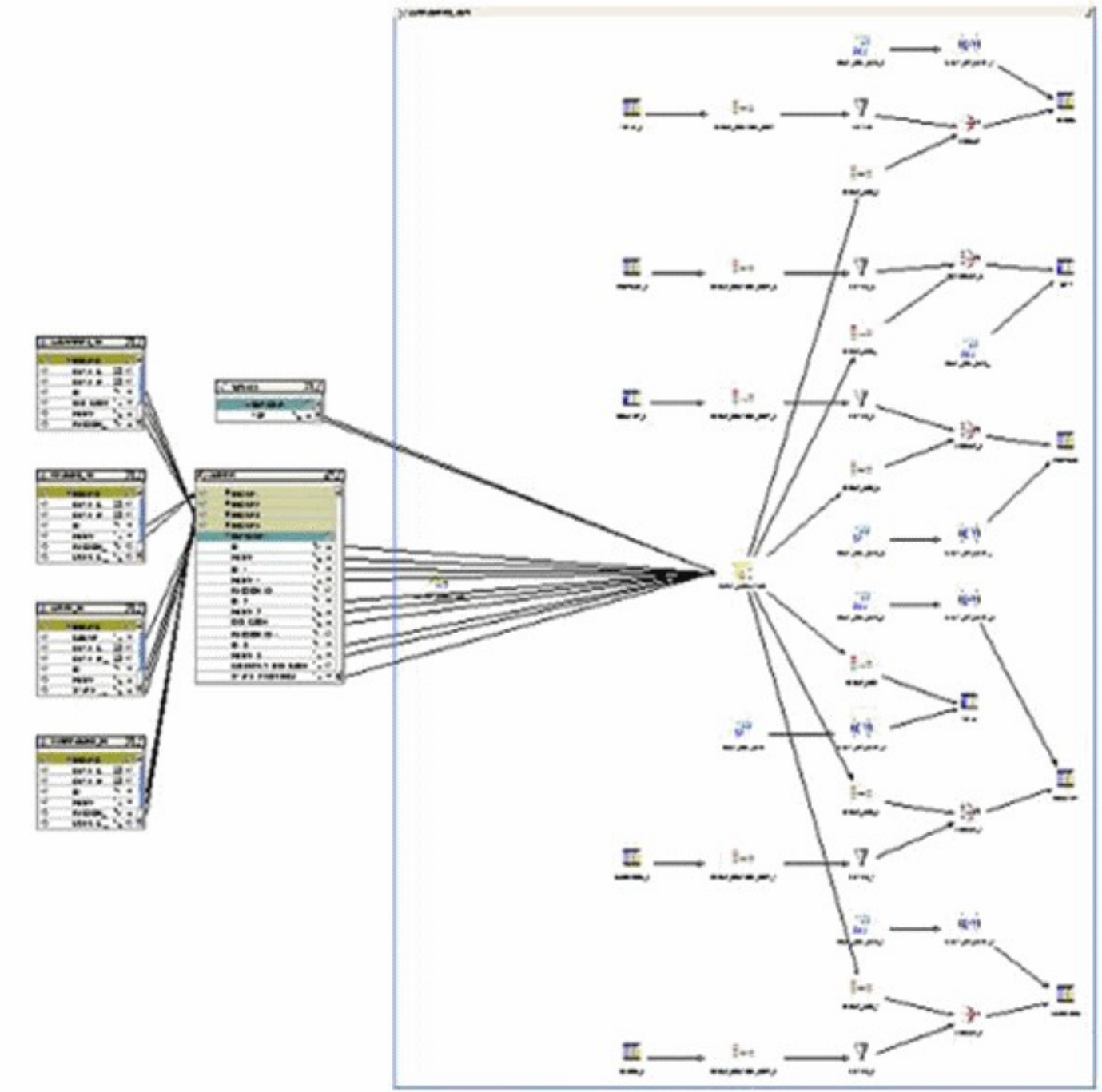
The ETL Process

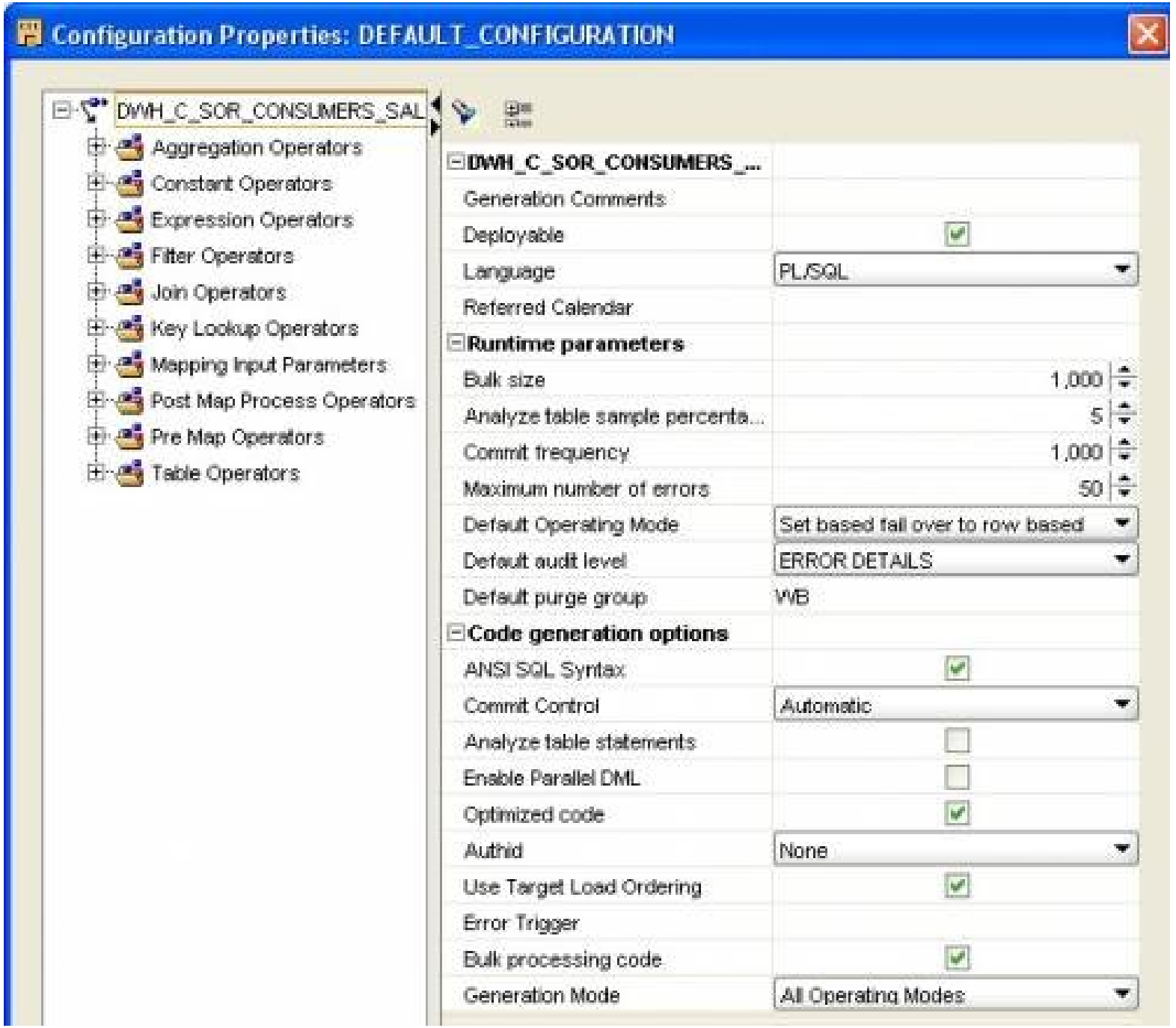
- Capture
- Scrub or data cleansing
- Transform
- Load and Index

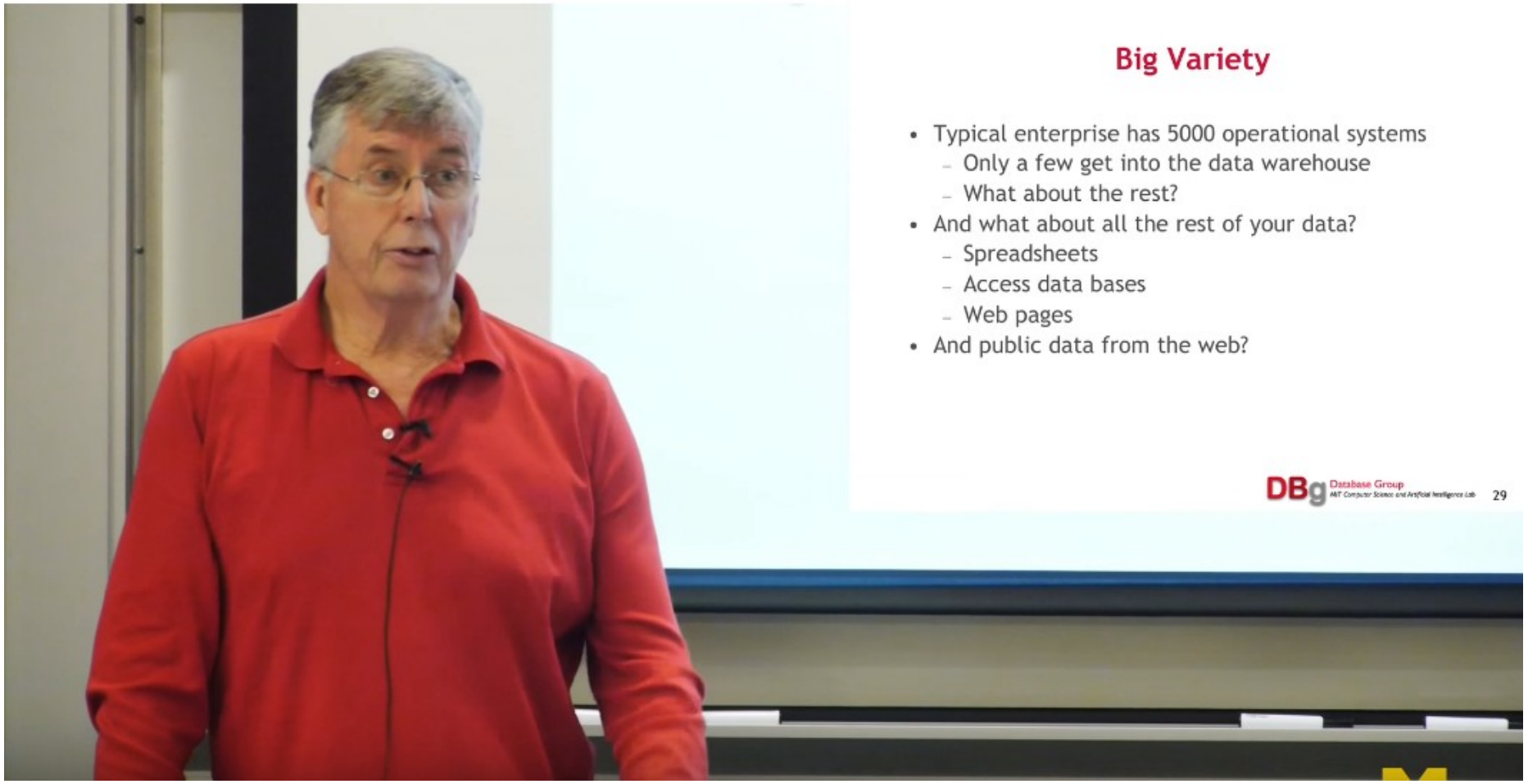
ETL = Extract, transform, and load

ETL Tools

- Declarative rules (what)
  - Mappings and transformations defined graphically
- Actual implementation (how)
  - Handling of errors
  - Performance
  - Auditing
  - Method
    - Also change data capture




Configuration Properties: DEFAULT_CONFIGURATION
DWH_C_SOR_CONSUMERS_SAL
Aggregation Operators
Constant Operators
Expression Operators
Filter Operators
Join Operators
Key Lookup Operators
Mapping Input Parameters
Post Map Process Operators
Pre Map Operators
Table Operators
DWH_C_SOR_CONSUMERS_...
Generation Comments
Deployable
Language
PL/SQL
Referred Calendar
Runtime parameters
Bulk size
1,000
Analyze table sample percenta...
5
Commit frequency
1,000
Maximum number of errors
50
Default Operating Mode
Set based fail over to row based
Default audit level
ERROR DETAILS
Default purge group
WB
Code generation options
ANSI SQL Syntax
Commit Control
Automatic
Analyze table statements
Enable Parallel DML
Optimized code
Authid
None
Use Target Load Ordering
Error Trigger
Bulk processing code
Generation Mode
All Operating Modes



https://youtu.be/KRcecxdGxvQ?t=2379

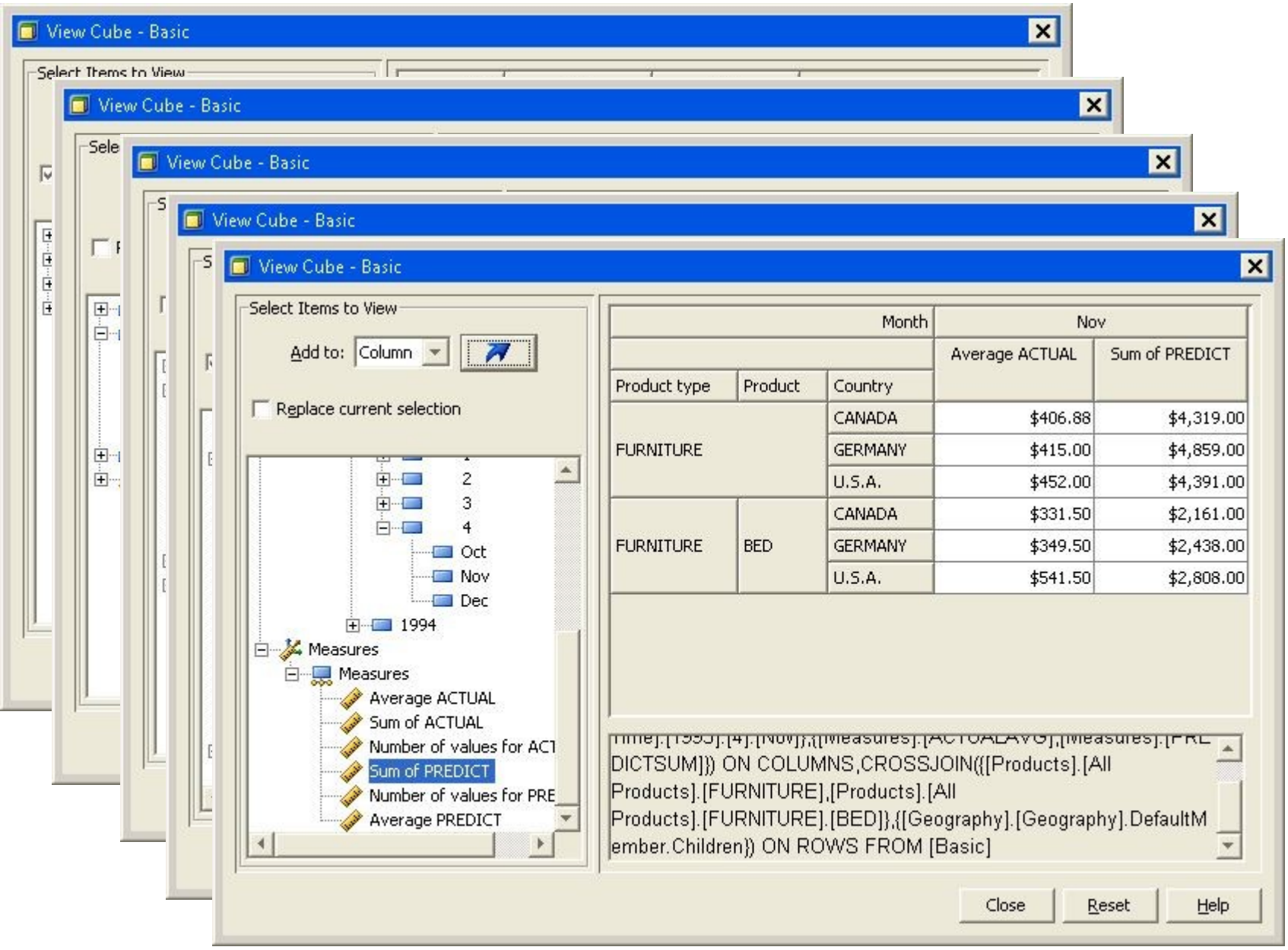
QUERYING TOOLS

(Querying Tools)

- SQL is not an analytical language
- SQL-99 includes some data warehousing extensions
- SQL-99 still is not a full-featured data warehouse querying and analysis tool.
- Different DBMS vendors will implement some or all of the SQL-99 OLAP extension commands and possibly others.

On-Line Analytical Processing (OLAP)

- OLAP is the use of a set of graphical tools that provides users with multidimensional views of their data and allows them to analyze the data using simple windowing techniques
- A data warehouse is based on a multidimensional data model which views data in the form of a data cube
- A data cube, such as *sales*, allows data to be modeled and viewed in multiple dimensions
  - Dimension tables, such as *item* (*item_name, brand, type*), or *time* (*day, week, month, quarter, year*)
  - Fact table contains measures (such as *dollars_sold*) and keys to each of the related dimension tables


View Cube - Basic
Select Items to View
Add to: Column
Replace current selection
2
3
4
Oct
Nov
Dec
1994
Measures
Measures
Average ACTUAL
Sum of ACTUAL
Number of values for ACT
Sum of PREDICT
Number of values for PRE
Average PREDICT
Month
Nov
Average ACTUAL
Sum of PREDICT
Product type
Product
Country
FURNITURE
CANADA
$406.88
$4,319.00
GERMANY
$415.00
$4,859.00
U.S.A.
$452.00
$4,391.00
FURNITURE
BED
CANADA
$331.50
$2,161.00
GERMANY
$349.50
$2,438.00
U.S.A.
$541.50
$2,808.00
DICTSUM]}) ON COLUMNS,CROSSJOIN({[Products].[All Products].[FURNITURE],[Products].[All Products].[FURNITURE].[BED]},{[Geography].[Geography].DefaultMember.Children}) ON ROWS FROM [Basic]
Close
Reset
Help

MOLAP Operations

- Roll up (drill-up): summarize data
  - *by climbing up hierarchy or by dimension reduction*
- Drill down (roll down): reverse of roll-up
  - *from higher level summary to lower level summary or detailed data, or introducing new dimensions*
- Slice and dice:
  - *project and select*

Figure 11-22: Slicing a data cube

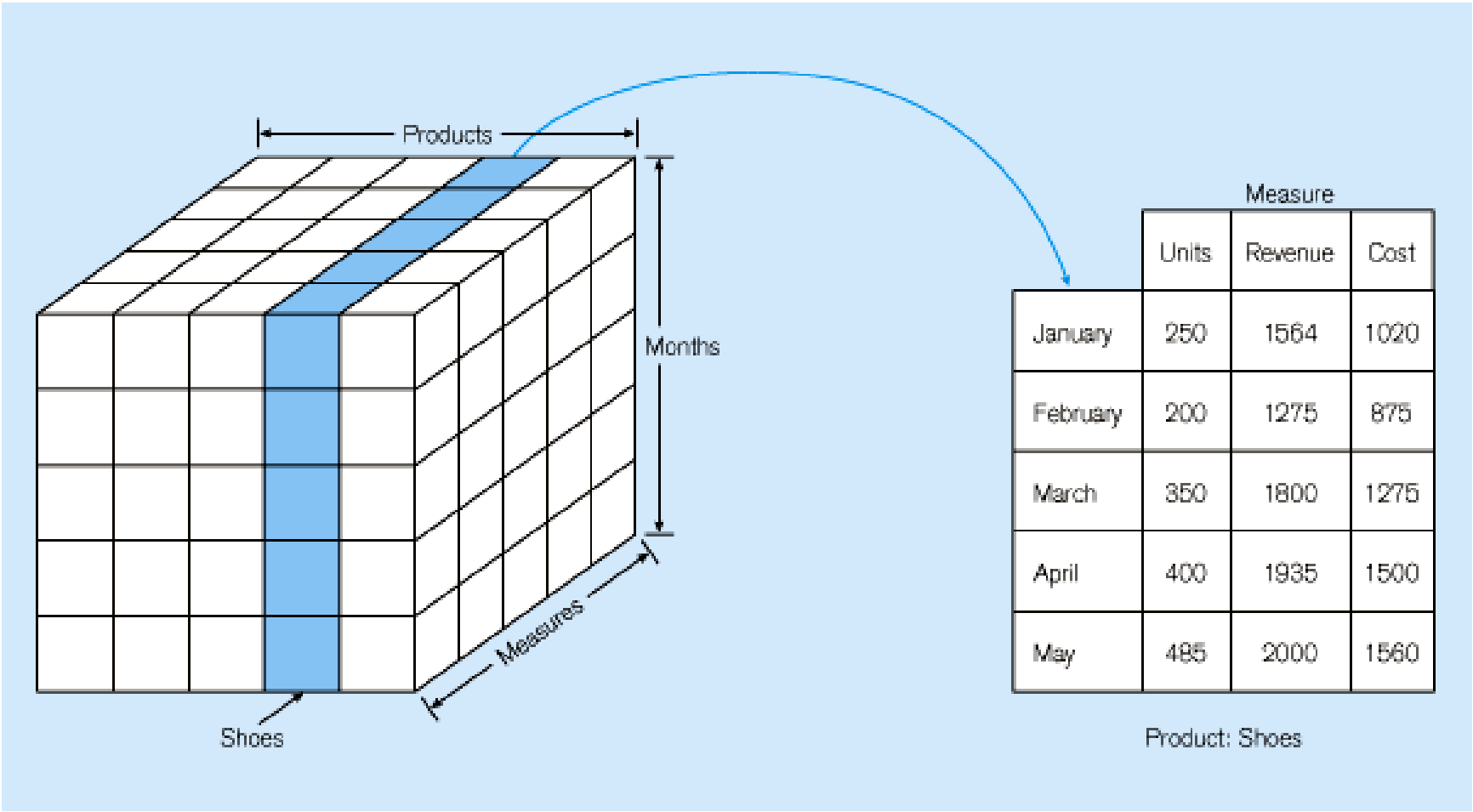


Figure 11-23:
Example of drill-down

Summary report

Brand	Package size	Sales
SofTowel	2-pack	$75
SofTowel	3-pack	$100
SofTowel	6-pack	$50

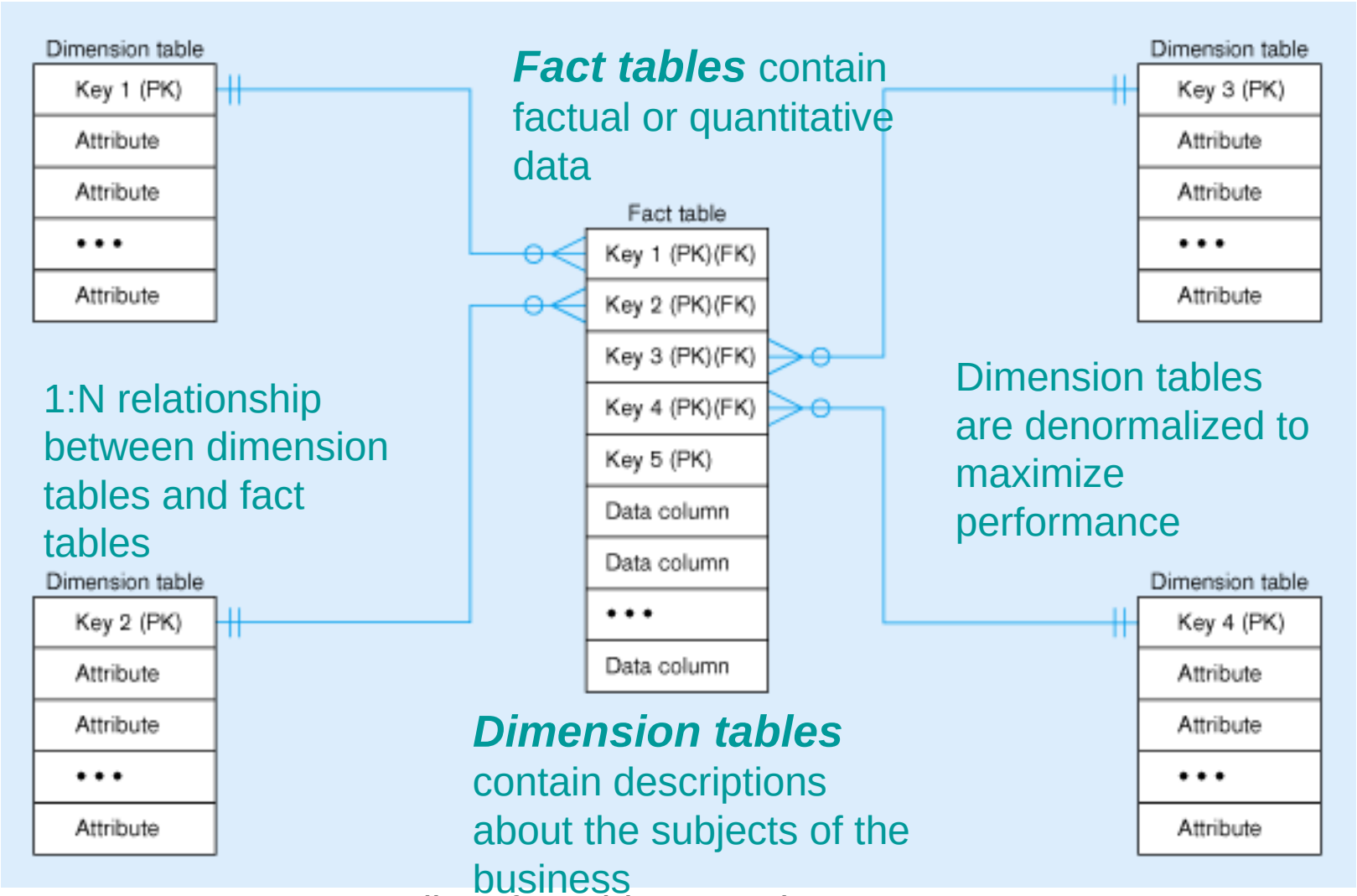
Drill-down with color added

Brand	Package size	Color	Sales
SofTowel	2-pack	White	$30
SofTowel	2-pack	Yellow	$25
SofTowel	2-pack	Pink	$20
SofTowel	3-pack	White	$50
SofTowel	3-pack	Green	$25
SofTowel	3-pack	Yellow	$25
SofTowel	6-pack	White	$30
SofTowel	6-pack	Yellow	$20

The Star Schema

- *Star schema*: is a simple database design in which dimensional (describing how data are commonly aggregated) are separated from fact or event data.
- A star schema consists of two types of tables: *fact tables* and *dimension table*.

Figure 11-13: Components of a **star schema**



Excellent for ad-hoc queries,
but bad for online transaction processing

Figure 11-14: Star schema example

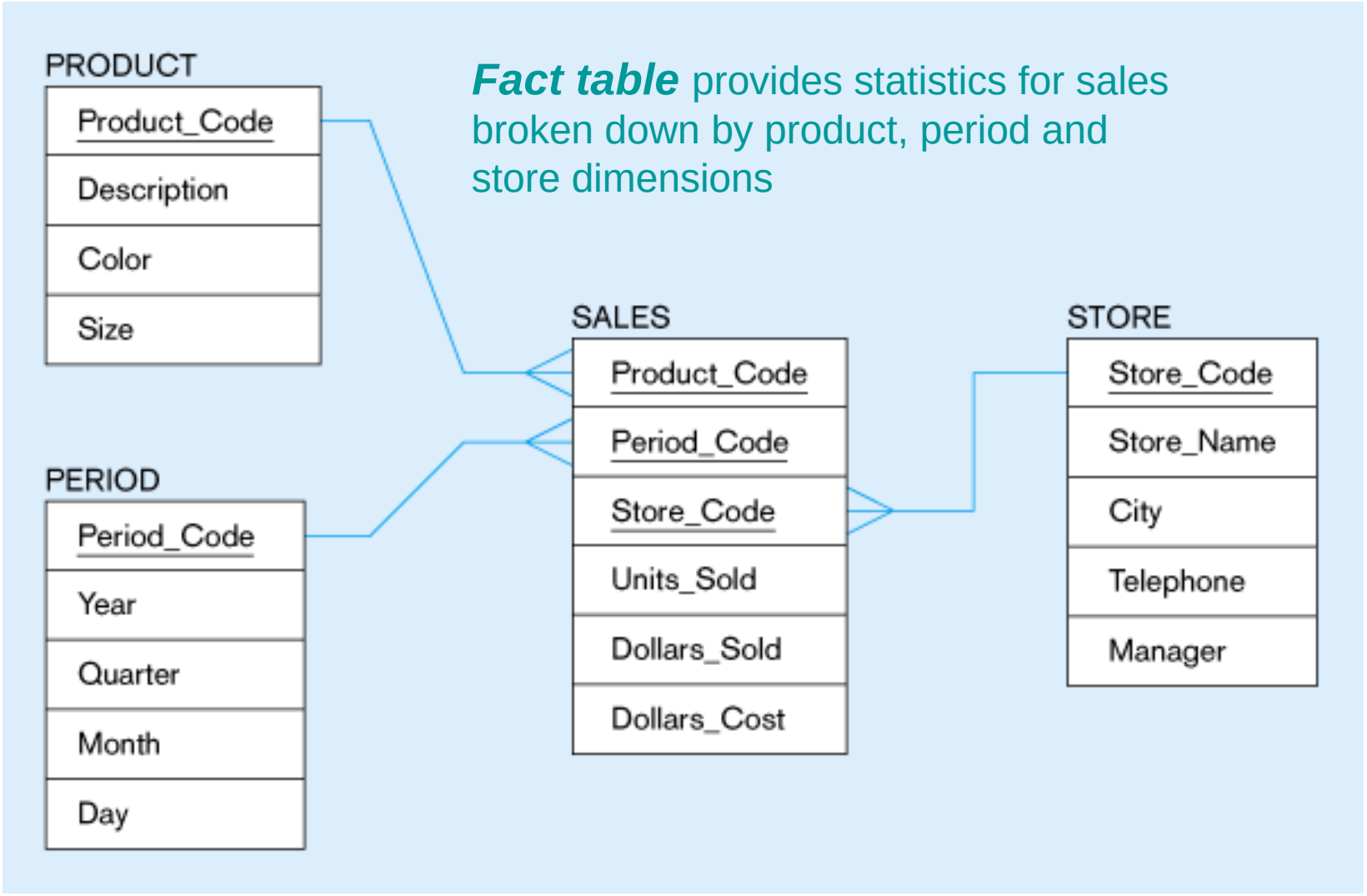
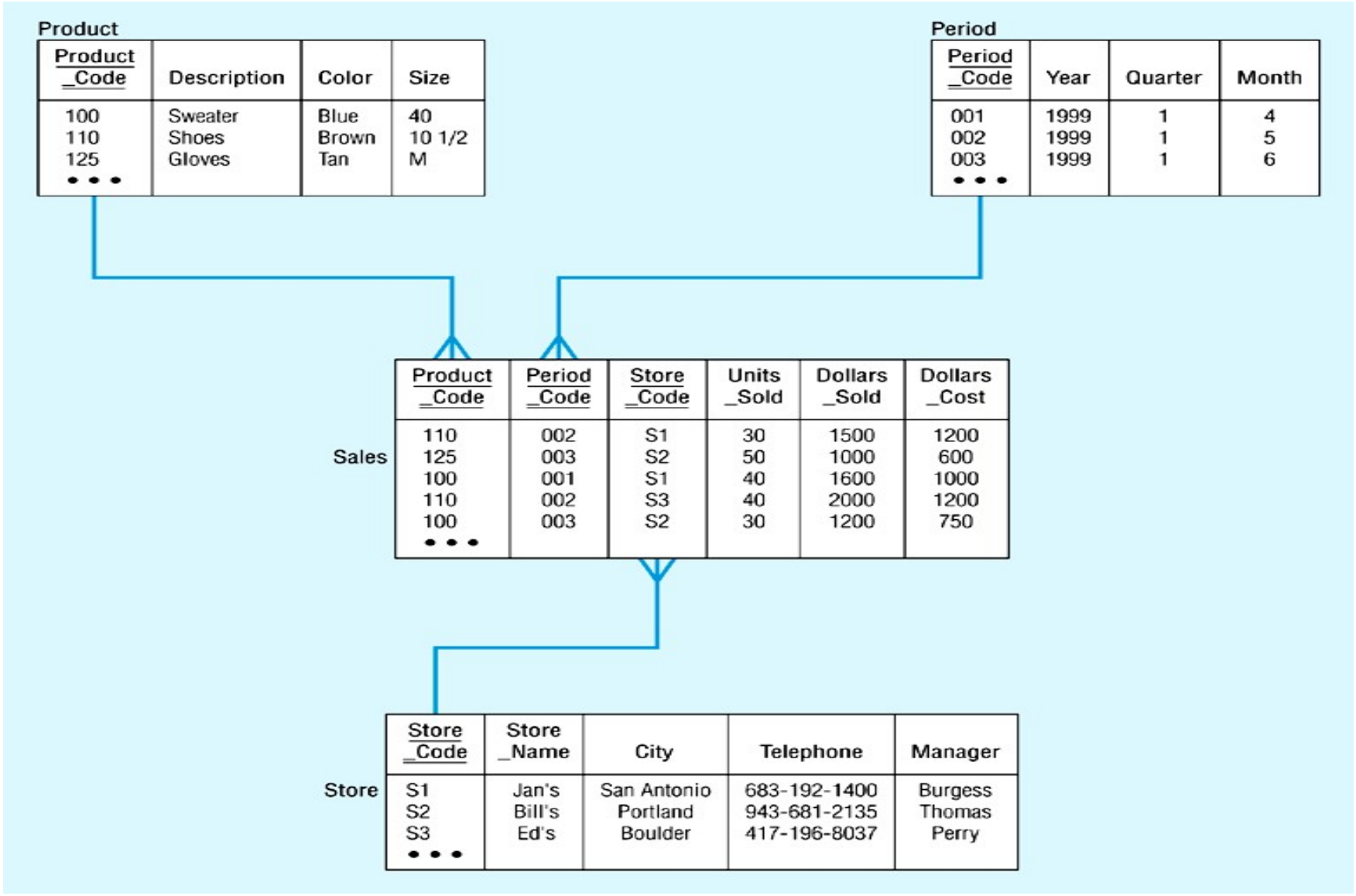


Figure 11-15: Star schema with sample data



(METADATA)

(Role of Metadata (data catalog))

- Identify subjects Identify dimensions and facts
- Indicate how data is derived from enterprise data warehouses, including derivation rules
- Indicate how data is derived from operational data store, including derivation rules
- Identify available reports and predefined queries
- Identify data analysis techniques (e.g. drill-down)
- Identify responsible people

DATA QUALITY

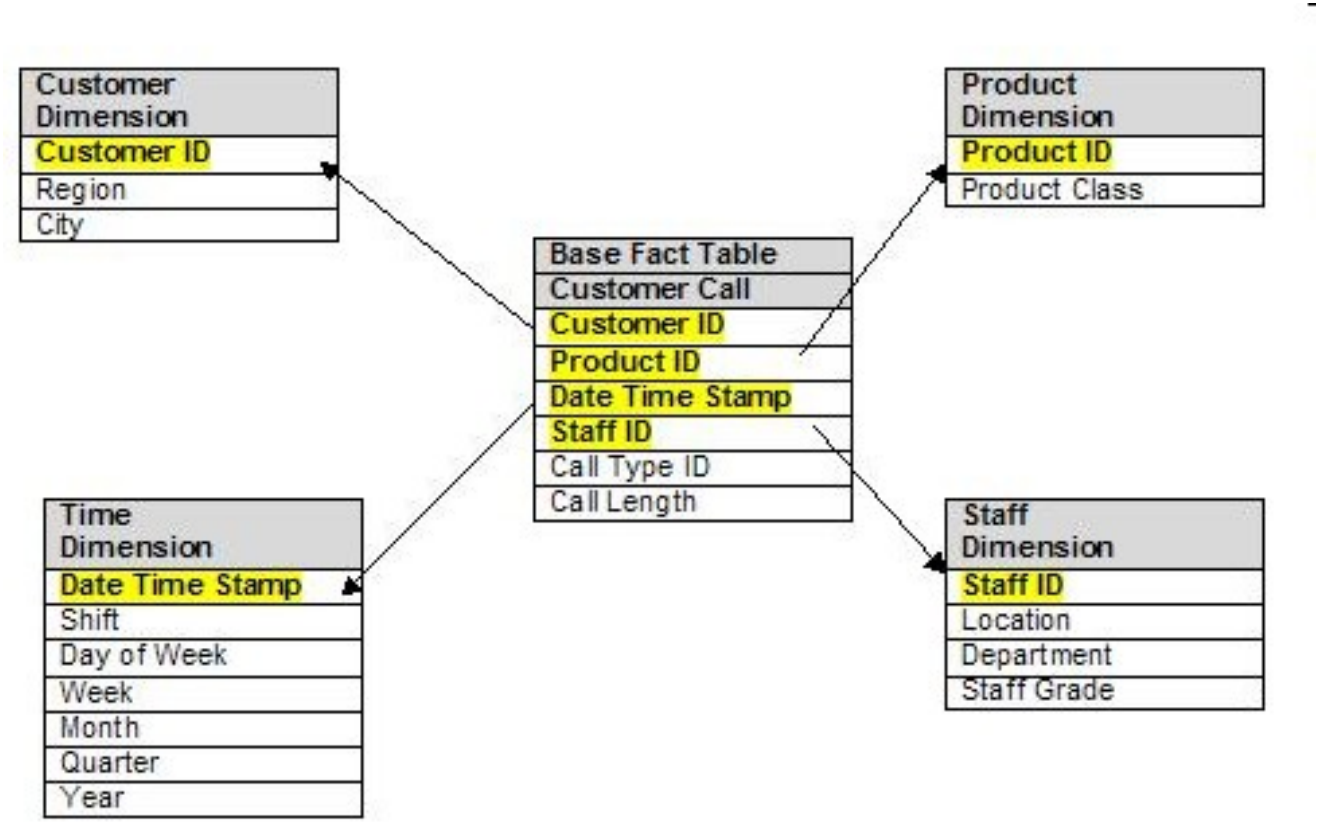
Optimisation

Data warehouse queries

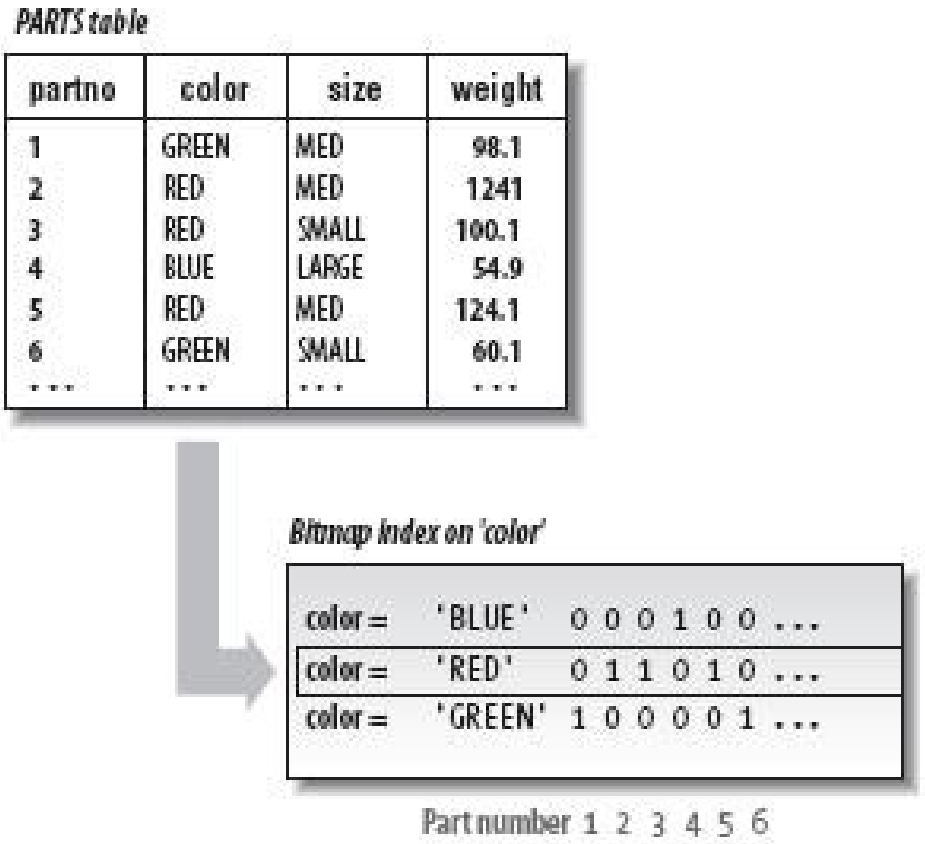
- Large number of records
- Ad-hoc queries, multiple dimensions
- Aggregations

- Read operations -

Star Schema (Bitmap index)



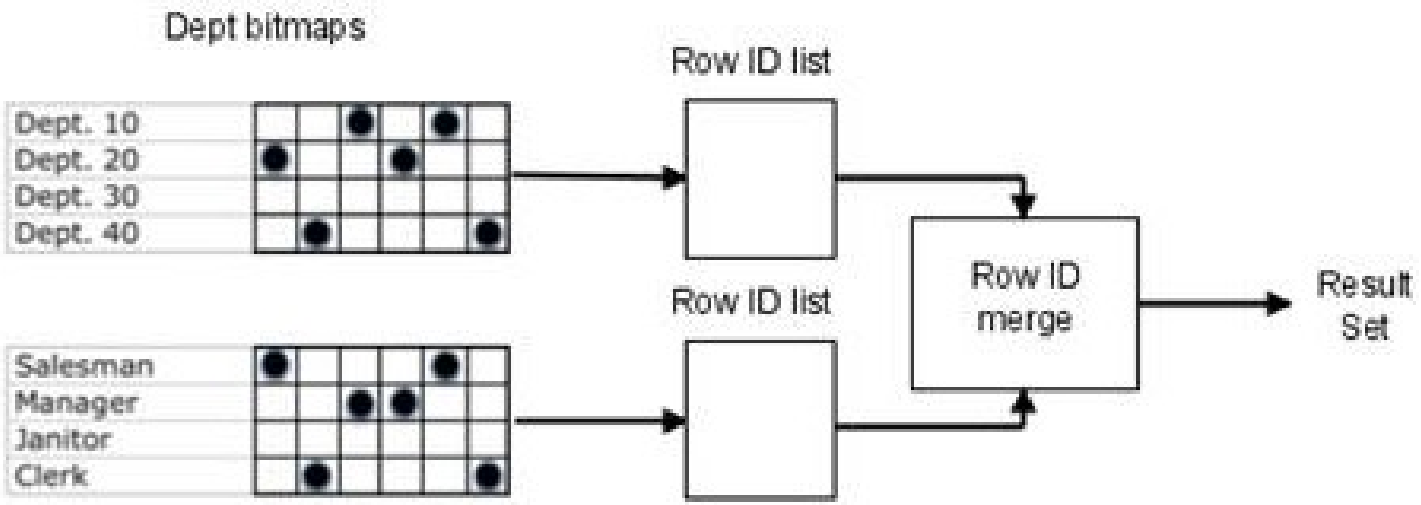
Bitmap index



+ compression

- **Low cardinality columns**
- **Infrequently updated or read-only tables**

Bitmap index



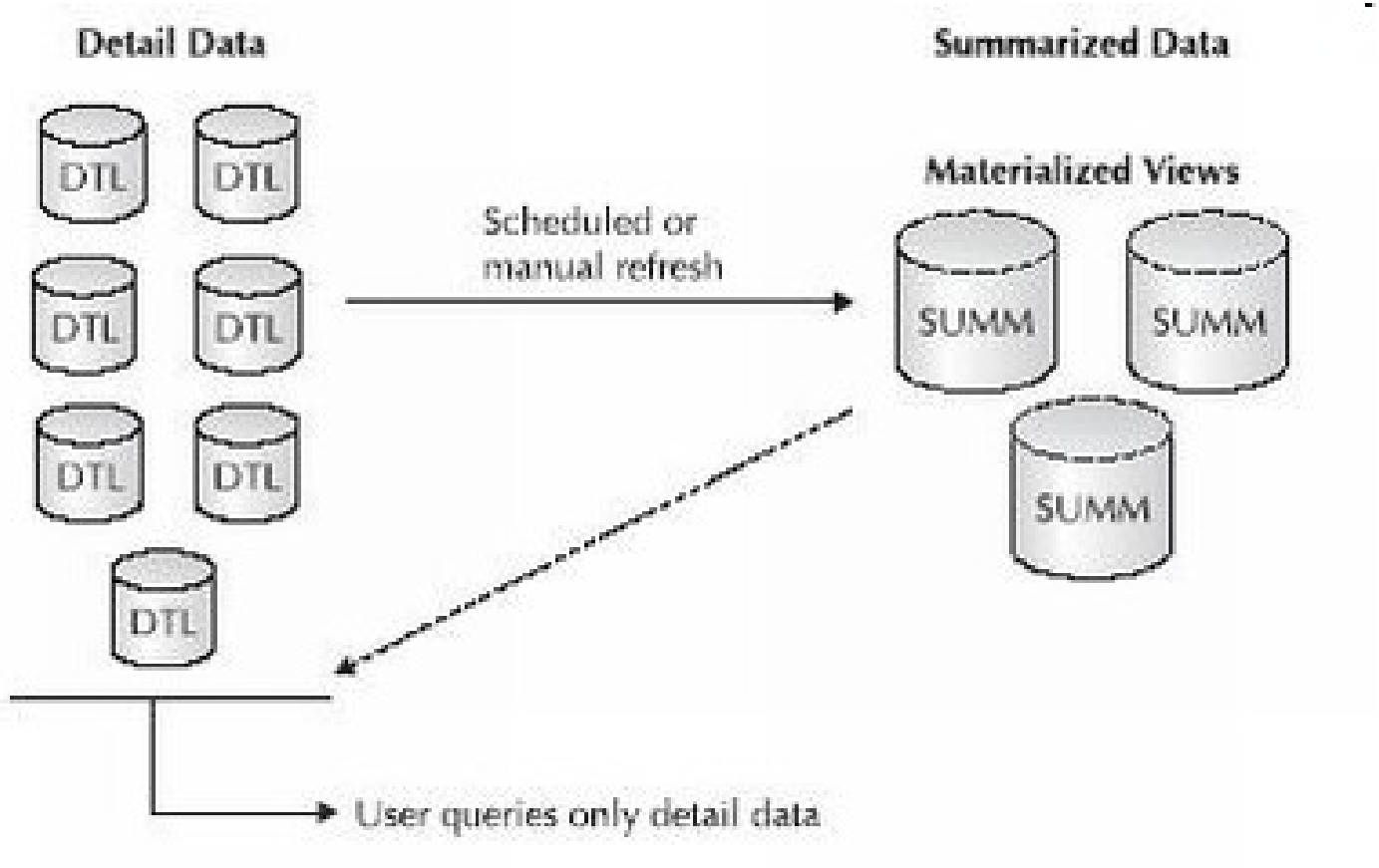
Bitmap index

status = 'married'		region = 'central'		region = 'west'						
0		0		0		0		0		0
1		1		0		1		1		1
1	AND	0	OR	1	=	1	AND	1	=	1
0		0		1		0		1		0
0		1		0		0		1		0
1		1		0		1		1		1

Star transformation

```
SELECT ch.channel_class, c.cust_city, t.calendar_quarter_des
FROM sales s, times t, customers c, channels ch
WHERE s.time_id = t.time_id
AND s.cust_id = c.cust_id
AND s.channel_id = ch.channel_id
AND c.cust_state_province = 'CA'
AND ch.channel_desc IN ('Internet','Catalog')
AND t.calendar_quarter_desc IN (‘2006-Q1',‘2006-Q2')

SELECT ch.channel_class, c.cust_city, t.calendar_quarter_desc
FROM sales WHERE
time_id IN
  (SELECT time_id FROM times WHERE calendar_quarter_desc
      IN(‘2006-Q1',‘2006-Q2'))
AND cust_id IN
  (SELECT cust_id FROM customers WHERE cust_state_province='CA'
AND channel_id IN
  (SELECT channel_id FROM channels WHERE channel_desc IN
      ('Internet','Catalog'));
```

Materialized view, query rewrite

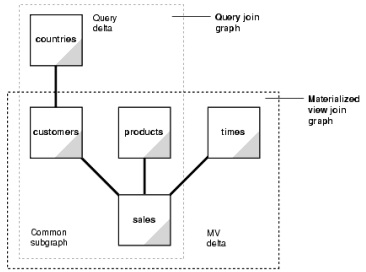


Materialized view

```
CREATE MATERIALIZED VIEW
  comm_prod_mv
  SELECT sales_rep_id, prod_id,
         comm_date, count(*),
         sum(comm_amt)
  FROM commission
  GROUP BY sales_rep_id,
           prod_id,  comm_date;
```

Materialized view, query rewrite

- Join Compatibility Check
  - Common joins that occur in both the query and the materialized view. These joins form the common subgraph.
  - Delta joins that occur in the query but not in the materialized view. These joins form the query delta subgraph.
  - Delta joins that occur in the materialized view but not in the query. These joins form the materialized view delta subgraph.
- Data Sufficiency Check
- Grouping Compatibility Check
- Aggregate Computability Check



Materialized view, query rewrite

Table 19-1 Dimension and Constraint Requirements for Query Rewrite

Query Rewrite Types	Dimensions	Primary Key/Foreign Key/Not Null Constraints
Matching SQL Text	Not Required	Not Required
Join Back	Required OR	Required
Aggregate Computability	Not Required	Not Required
Aggregate Rollup	Not Required	Not Required
Rollup Using a Dimension	Required	Not Required
Filtering the Data	Not Required	Not Required
PCT Rewrite	Not Required	Not Required
Multiple Materialized Views	Not Required	Not Required

Query AVG(X) MVI: SUM(X) and COUNT(

Query: years MVI: months

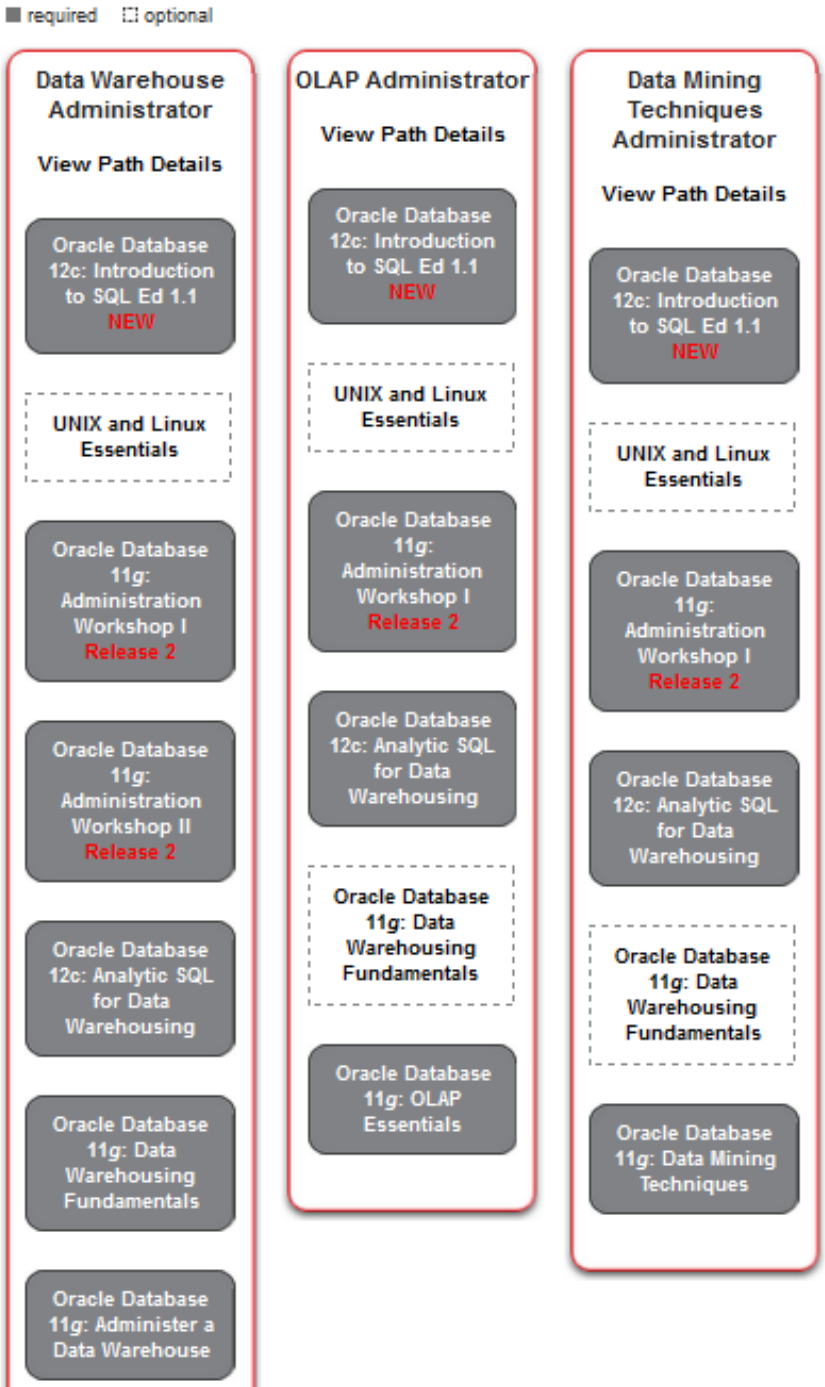
Refresh modes, query rewrite integrity

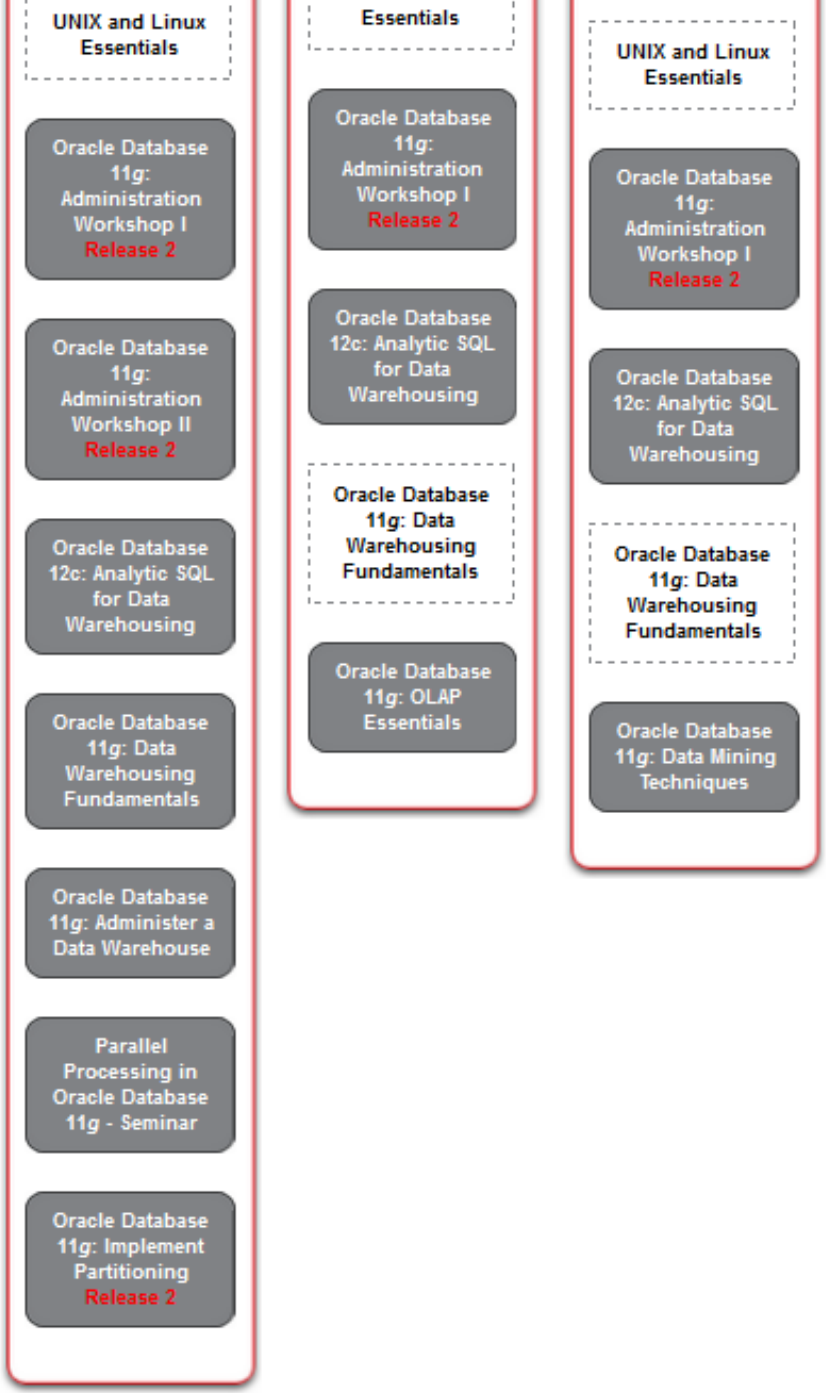
- Refresh types
  - Complete
  - Fast
- Accuracy of Query Rewrite (Oracle)
  - Enforced
  - Trusted
  - Stale_tolerated

További irodalom

- **Oracle® Database Data Warehousing Guide 11*g* Release 2 (11.2)**
  - http://docs.oracle.com/cd/E11882_01/server.112/e25554/toc.htm
  - **Basic Query Rewrite - http://docs.oracle.com/cd/E11882_01/server.112/e25554/qrbasic.htm#DWHSG018**
  - **Advanced Query Rewrite - http://docs.oracle.com/cd/E11882_01/server.112/e25554/qradv.htm#DWHSG080**
- **Oracle® Database 2 Day + Data Warehousing Guide 11*g* Release 2 (11.2)**
  - http://docs.oracle.com/cd/E11882_01/server.112/e25555/toc.htm
- **Data Warehousing and Business Intelligence**
  - http://docs.oracle.com/cd/E11882_01/nav/portal_6.htm

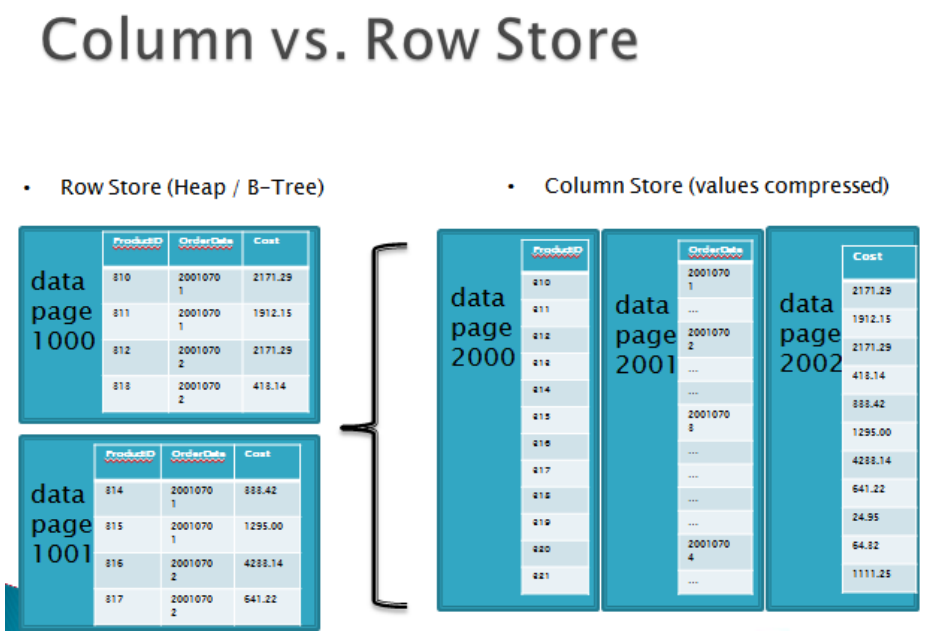
- Gajdos Sándor: Adattárház alapú vezet ői információs rendszerek https://db.bme.hu/~gajdos/2012adatb2/1.%20eloadas%20Analitikus%20alkalmazasok%20ppt.pdf
- Gajdos Sándor: Adattárházak és alkalmazásaik https://www.db.bme.hu/targyak/adattarhazak-alkalmazasaik

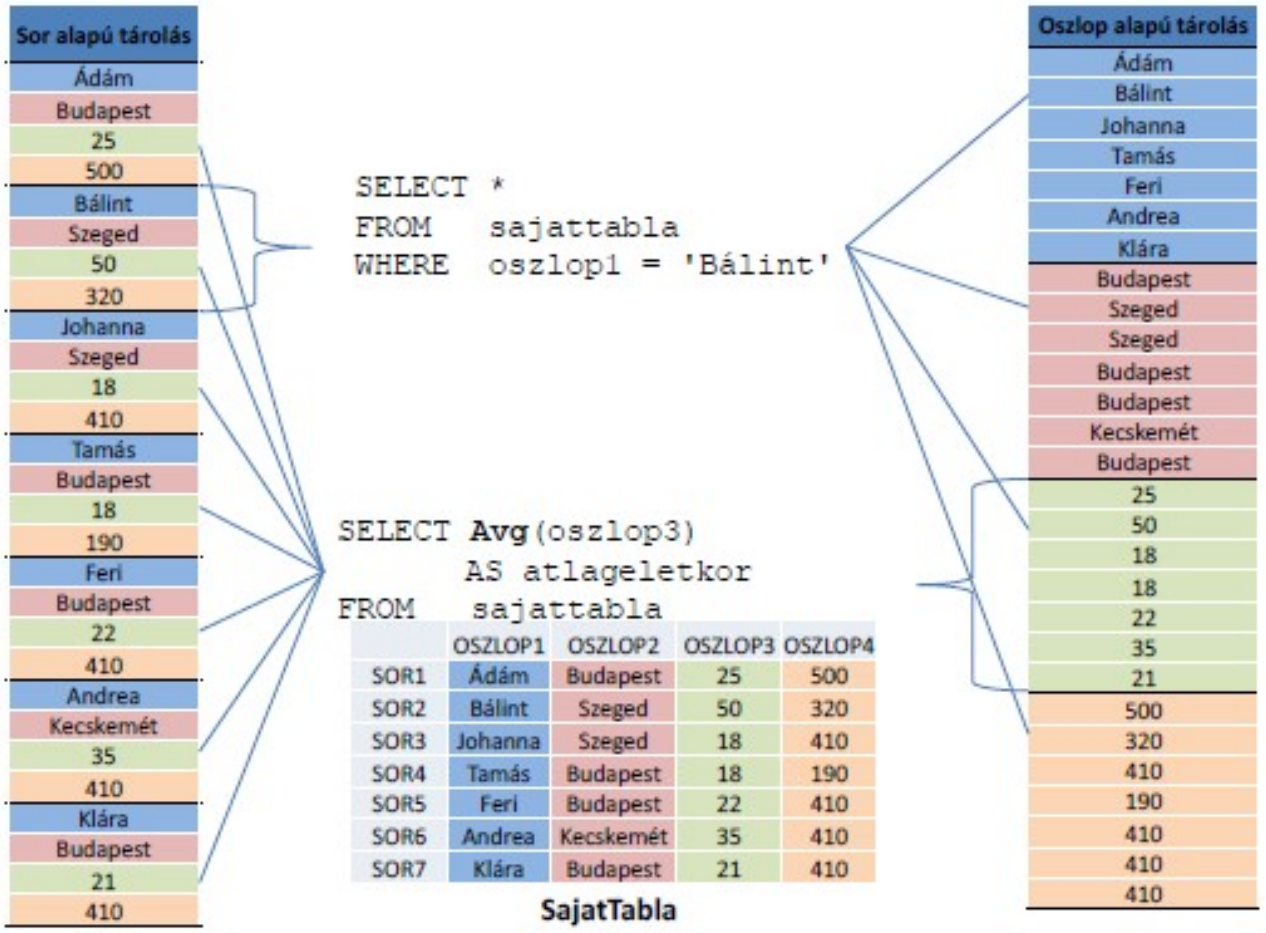

required
optional
Data Warehouse Administrator
View Path Details
Oracle Database 12c: Introduction to SQL Ed 1.1 NEW
UNIX and Linux Essentials
Oracle Database 11g: Administration Workshop I Release 2
Oracle Database 11g: Administration Workshop II Release 2
Oracle Database 12c: Analytic SQL for Data Warehousing
Oracle Database 11g: Data Warehousing Fundamentals
Oracle Database 11g: Administer a Data Warehouse
OLAP Administrator
View Path Details
Oracle Database 12c: Introduction to SQL Ed 1.1 NEW
UNIX and Linux Essentials
Oracle Database 11g: Administration Workshop I Release 2
Oracle Database 12c: Analytic SQL for Data Warehousing
Oracle Database 11g: Data Warehousing Fundamentals
Oracle Database 11g: OLAP Essentials
Data Mining Techniques Administrator
View Path Details
Oracle Database 12c: Introduction to SQL Ed 1.1 NEW
UNIX and Linux Essentials
Oracle Database 11g: Administration Workshop I Release 2
Oracle Database 12c: Analytic SQL for Data Warehousing
Oracle Database 11g: Data Warehousing Fundamentals
Oracle Database 11g: Data Mining Techniques


UNIX and Linux Essentials
Oracle Database 11g: Administration Workshop I Release 2
Oracle Database 11g: Administration Workshop II Release 2
Oracle Database 12c: Analytic SQL for Data Warehousing
Oracle Database 11g: Data Warehousing Fundamentals
Oracle Database 11g: Administer a Data Warehouse
Parallel Processing in Oracle Database 11g - Seminar
Oracle Database 11g: Implement Partitioning Release 2
Essentials
Oracle Database 11g: Administration Workshop I Release 2
Oracle Database 12c: Analytic SQL for Data Warehousing
Oracle Database 11g: Data Warehousing Fundamentals
Oracle Database 11g: OLAP Essentials
UNIX and Linux Essentials
Oracle Database 11g: Administration Workshop I Release 2
Oracle Database 12c: Analytic SQL for Data Warehousing
Oracle Database 11g: Data Warehousing Fundamentals
Oracle Database 11g: Data Mining Techniques

Column-oriented DBMS

- Column-oriented
  - Aggregates over many rows and few columns
  - Compression…
- Row-oriented
  - Retrieving or changing few records with many attributes





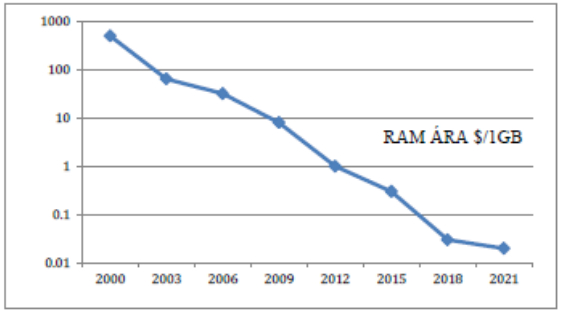
	OSZLOP1	OSZLOP2	OSZLOP3	OSZLOP4
SOR1	Ádám	Budapest	25	500
SOR2	Bálint	Szeged	50	320
SOR3	Johanna	Szeged	18	410
SOR4	Tamás	Budapest	18	190
SOR5	Feri	Budapest	22	410
SOR6	Andrea	Kecskemét	35	410
SOR7	Klára	Budapest	21	410

(Sípos Zsófia, szakdolgozat)

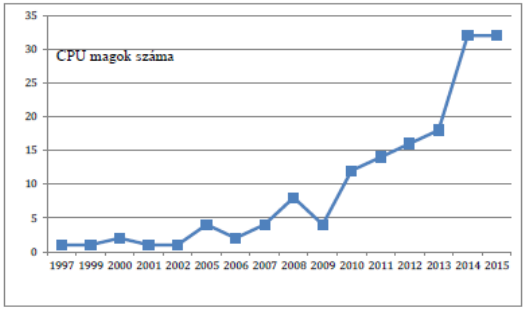
In-memory databases

(részben Sípos Zsófi szakdolgozatából)

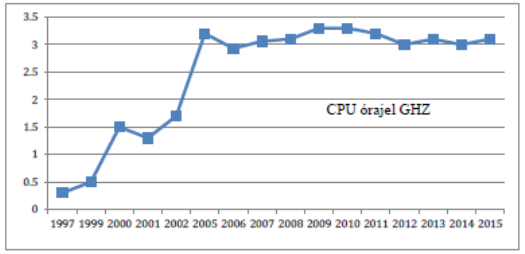
- RAM: szerver 1-6 TB !
- CPU magok száma, frekvencia
- SSD
  - (Logfájlok is!)



3. ábra A RAM árának csökkenése az elmúlt és a közeljövő években $/1GB-ban



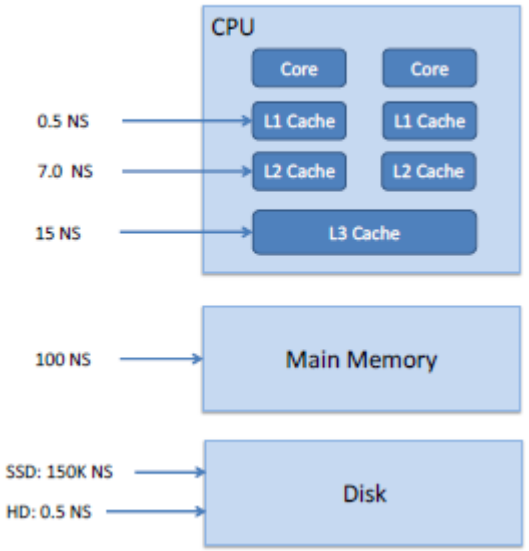
1. ábra a CPU magjainak számának növekedése az elmúlt években

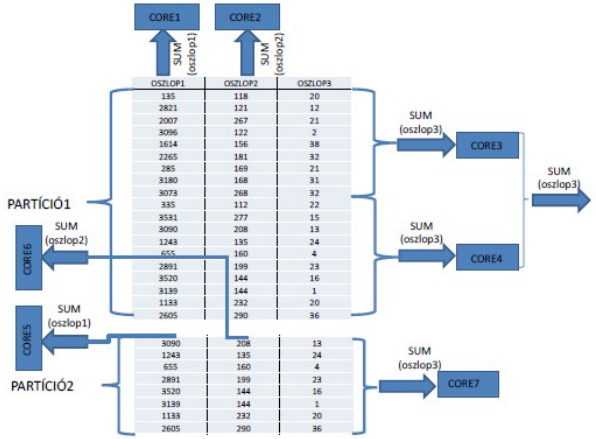


2. ábra a CPU órajelének növekedése az elmúlt években GHZ-ben

SAP HANA (High Speed Analytic Appliance)

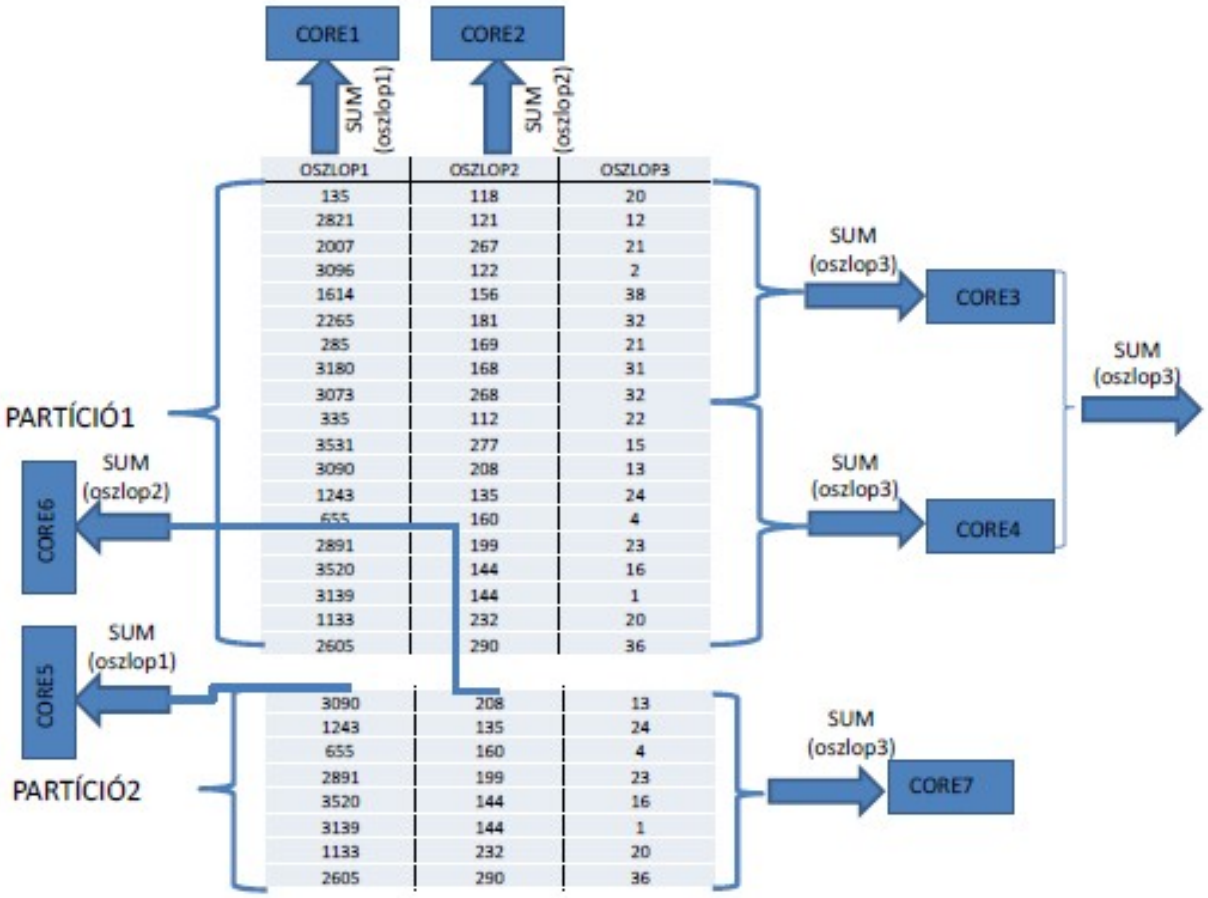
- Oszlop alapú tárolás is
  - Memóriából is gyorsabb szekvenciálisan olvasni
  - Tömörítés
- Párhuzamosítás





10. ábra A vertikális, horizontális és oszlopon belüli párhuzamosítás

SAP HANA
(High Speed Analytic Appliance)



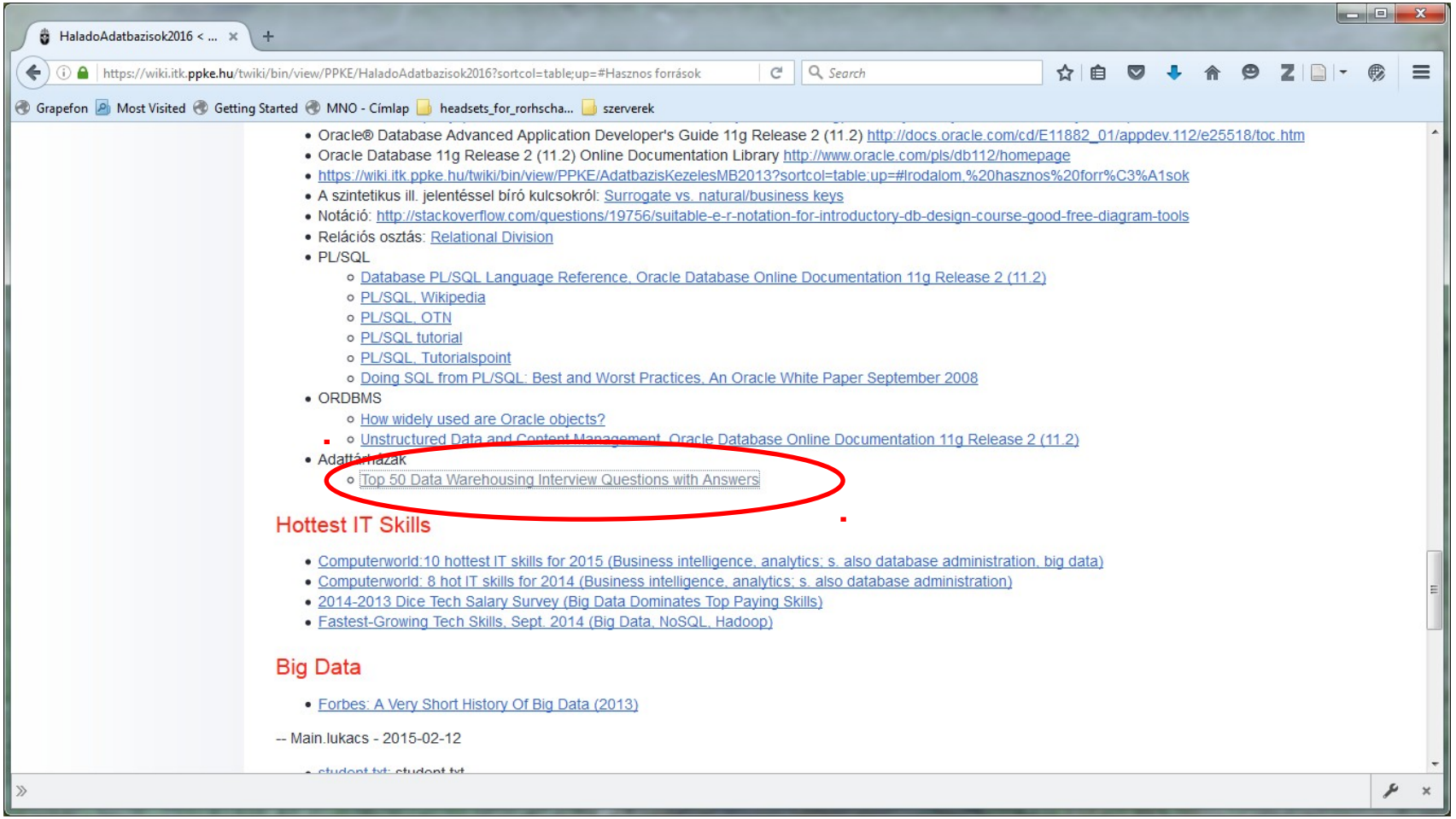
10. ábra A vertikális, horizontális és oszlopon belüli párhuzamosítás

In-memory adatbáziskezelés, adattárház?

- Kb. 2 nagyságrenddel gyorsabb adatfeldolgozás (pl. 6 óra -> 6 perc)
- OLTP és OLAP együtt!

- Oracle Database In-Memory
  - http://www.oracle.com/technetwork/database/in-memory/overview/twp-oracle-database-in-memory-2245633.html

Top 50 Data Warehousing Interview Questions with Answers



A következő materializált nézetek vannak **elkészítve** az adatbázisban (minden a ad18___db sémában) -- csak tájékoztatásul:

```
CREATE materialized VIEW historyitems_large_mv2 ENABLE QUERY REWRITE
AS
  SELECT user_id,
    SUM(duration),
    COUNT(duration)
  FROM historyitems_large
  GROUP BY user_id;


CREATE materialized VIEW historyitems_large_mv3 ENABLE QUERY REWRITE
AS
  SELECT user_id,
    TO_CHAR(started_at, 'YYYY-MM'),
    SUM(duration),
    COUNT(duration)
  FROM lukacs.historyitems_large
  GROUP BY user_id,
    TO_CHAR(started_at, 'YYYY-MM');


CREATE materialized VIEW historyitems_large_mv4 ENABLE QUERY REWRITE
AS
  SELECT user_id,
    TO_CHAR(started_at, 'YYYY'),
    SUM(duration),
    COUNT(duration)
  FROM ad18___db.historyitems_large
  GROUP BY user_id,
    TO_CHAR(started_at, 'YYYY');
```

Nézzük meg a következő lekérdezések lekérdezési tervét és lekérdezési költségét így és a lekérdezés újraírás letiltásával (/*+ NOREWRITE */ hint)

```
SELECT user_id, AVG(duration) FROM ad18___db.historyitems_large GROUP BY user_id;

SELECT DISTINCT user_id FROM ad18___db.historyitems_large;

SELECT SUBSTR(users.surname,8,1),
  SUM(historyitems_large.duration)
FROM ad18___db.historyitems_large
LEFT OUTER JOIN ad18___db.users
ON historyitems_large.user_id = users.id
GROUP BY SUBSTR(users.surname,8,1)
ORDER BY SUM(historyitems_large.duration) DESC;

SELECT user_id,
  TO_CHAR(started_at, 'YYYY'),
  AVG(duration)
FROM ad18___db.historyitems_large
GROUP BY user_id,
  TO_CHAR(started_at, 'YYYY');
```

A Query rewrite mechanizmus meglehetősen bonyolult, sokszor nehéz látni, miért nem sikerült az Oracle-nek a lekérdezést újraírnia.
Ezért biztosít egy eljárást, aminek egy lekérdezést és egy materializált nézet nevet string-ként átadva információt ad a lekérdezés újraírásról, ill. az újraírás akadályáról. Az információkat egy fix sémájú, rewrite_table nevű táblába írja.

Sajnos jogosultság hiányában nem tudják maguk végrehajtani, de itt egy példa végrehajtás:

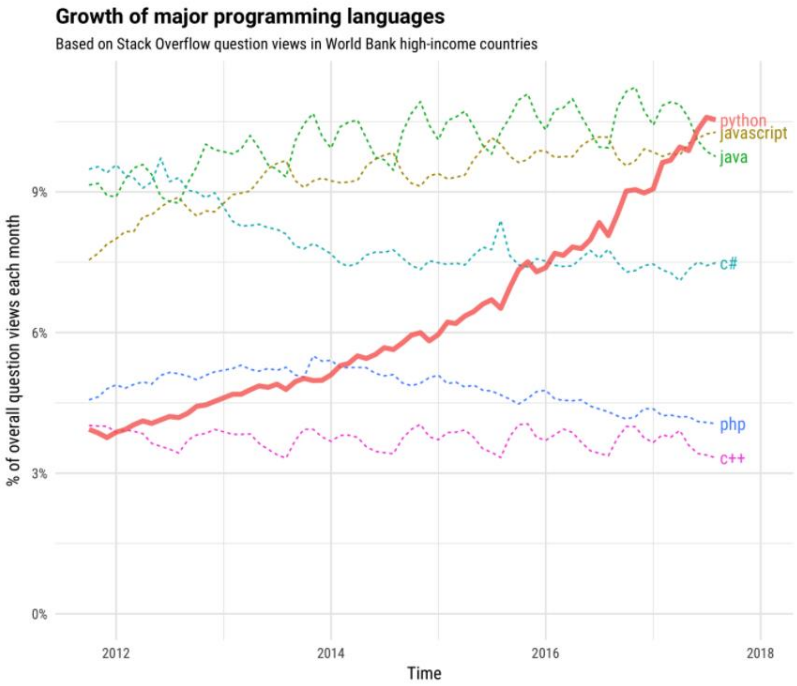
```
BEGIN
     DBMS_MVIEW.EXPLAIN_REWRITE('select historyitems_large.user_id, sum(historyitems_large.duration),
sum(audio_large.duration)
from historyitems_large left outer join AUDIO_LARGE on historyitems_large.audio_id = audio_large.id
group by historyitems_large.user_id','HISTORYITEMS_LARGE_MV2','100');
end;
/

select message from rewrite_table;

QSM-01150: a lekérdezés újraírása nem sikerült
QSM-01067: a(z) HISTORYITEMS_LARGE_MV2 statikus nézet nem támogatja a(z) SUM lekérdezés-mérőszámot
QSM-01082: A(z) HISTORYITEMS_LARGE_MV2 statikus nézet összekapcsolása a(z) HISTORYITEMS_LARGE táblával
nem lehetséges
QSM-01102: a(z) HISTORYITEMS_LARGE_MV2 statikus nézethez visszakapcsolás szükséges a(z)
HISTORYITEMS_LARGE táblához a(z) AUDIO_ID oszlopban
QSM-01219: nem található megfelelő statikus nézet a lekérdezés újraírásához
```

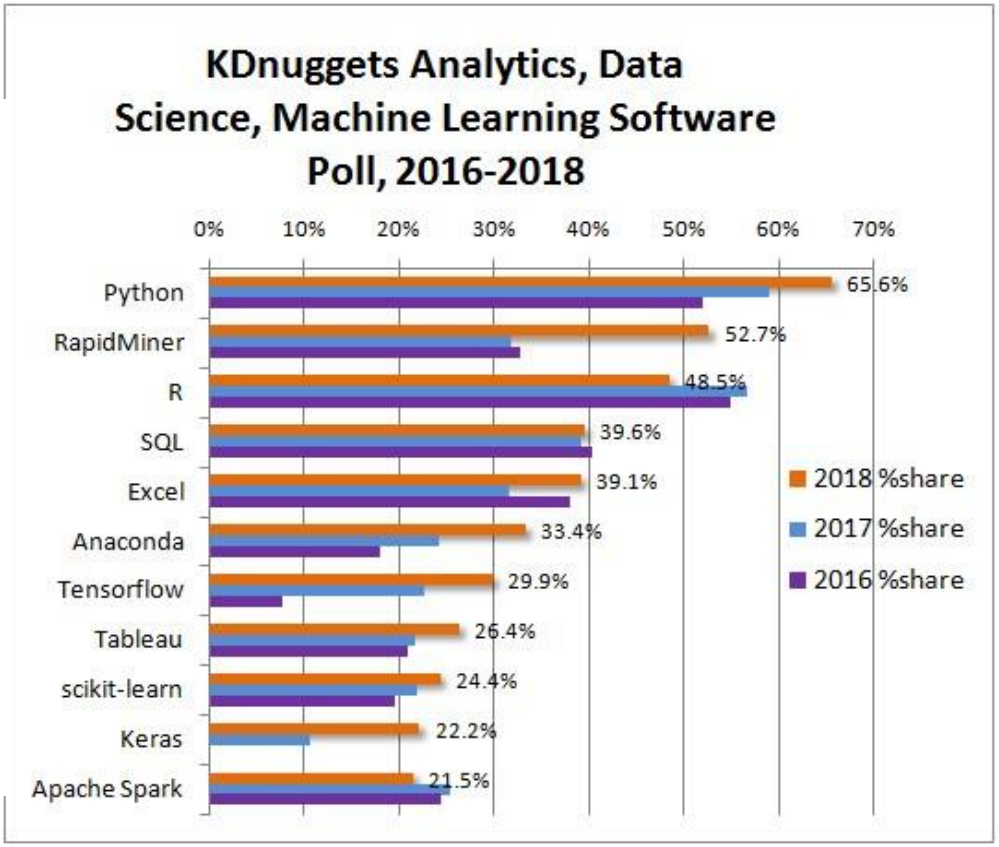
Database management II.

Python – Pandas library

Gergely Lukács
Pázmány Péter Catholic University
Faculty of Information Technology
Budapest, Hungary
lukacs@itk.ppke.hu

KDnuggets Analytics/Data Science 2018 Software Poll: top tools in 2018, and their share in the 2016-7 polls





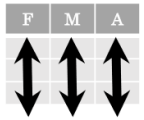
https://www.kdnuggets.com/2018/05/poll-tools-analytics-data-science-machine-learning-results.html

Pandas: Python Data Analysis Library Dataframes

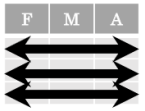
Data Wrangling with pandas Cheat Sheet http://pandas.pydata.org

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:



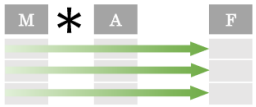
&



Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.



M * A

Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```

Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
          names=['n','v']))
```

Create DataFrame with a MultiIndex

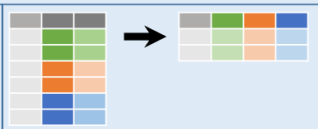
Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
    )
```

Reshaping Data – Change the layout of a data set

pd.melt(df)
Gather columns into rows.



df.pivot(columns='var', values='val')
Spread rows into columns.

pd.concat([df1,df2])
Append rows of DataFrames

pd.concat([df1,df2], axis=1)
Append columns of DataFrames

df.sort_values('mpg')
Order rows by values of a column (low to high).

df.sort_values('mpg',ascending=False)
Order rows by values of a column (high to low).

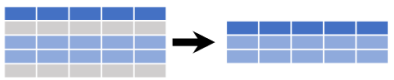
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df.sort_index()
Sort the index of a DataFrame

df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df.drop(columns=['Length','Height'])
Drop columns from DataFrame

Subset Observations (Rows)



df[df.Length > 7]
Extract rows that meet logical criteria.

df.drop_duplicates()
Remove duplicate rows (only considers columns).

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

df.sample(frac=0.5)
Randomly select fraction of rows.

df.sample(n=10)
Randomly select n rows.

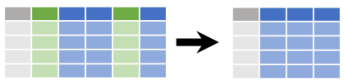
df.iloc[10:20]
Select rows by position.

df.nlargest(n, 'value')
Select and order top n entries.

df.nsmallest(n, 'value')
Select and order bottom n entries.

Logic in Python (and pandas)			
<	Less than	!=	Not equal to
>	Greater than	df.column.isin(*values*)	Group membership
==	Equals	pd.isnull(*obj*)	Is NaN
<=	Less than or equals	pd.notnull(*obj*)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

Subset Variables (Columns)



df[['width','length','species']]
Select multiple columns with specific names.

df['width'] *or* **df.width**
Select single column with specific name.

df.filter(regex='*regex*')
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples	
'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species$).*'	Matches strings except the string 'Species'

df.loc[:,'x2':'x4']
Select all columns between x2 and x4 (inclusive).

df.iloc[:,[1,2,5]]
Select columns in positions 1, 2 and 5 (first column is 0).

df.loc[df['a'] > 10, ['a','c']]
Select rows meeting logical condition, and only the specific columns .

Summarize Data

`df['w'].value_counts()`
Count number of rows with each unique value of variable

`len(df)`
# of rows in DataFrame.

`df['w'].nunique()`
# of distinct values in a column.

`df.describe()`
Basic descriptive statistics for each column (or GroupBy)

pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()`
Sum values of each object.

`count()`
Count non-NA/null values of each object.

`median()`
Median value of each object.

`quantile([0.25,0.75])`
Quantiles of each object.

`apply(function)`
Apply function to each object.

`min()`
Minimum value in each object.

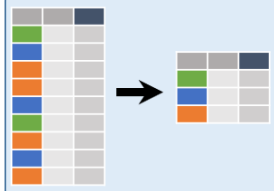
`max()`
Maximum value in each object.

`mean()`
Mean value of each object.

`var()`
Variance of each object.

`std()`
Standard deviation of each object.

Group Data



`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

`size()`
Size of each group.

`agg(function)`
Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

`shift(1)`
Copy with values shifted by 1.

`rank(method='dense')`
Ranks with no gaps.

`rank(method='min')`
Ranks. Ties get min rank.

`rank(pct=True)`
Ranks rescaled to interval [0, 1].

`rank(method='first')`
Ranks. Ties go to first value.

`shift(-1)`
Copy with values lagged by 1.

`cumsum()`
Cumulative sum.

`cummax()`
Cumulative max.

`cummin()`
Cumulative min.

`cumprod()`
Cumulative product.

Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.

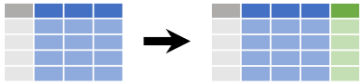
`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.

`df.fillna(value)`
Replace all NA/null data with value.

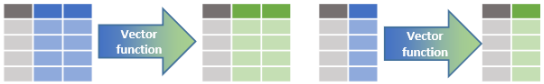
Make New Columns



`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.

`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.

`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.



pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

`max(axis=1)`
Element-wise max.

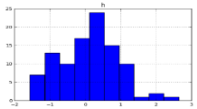
`clip(lower=-10,upper=10)`
Trim values at input thresholds

`min(axis=1)`
Element-wise min.

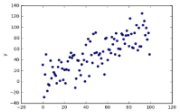
`abs()`
Absolute value.

Plotting

`df.plot.hist()`
Histogram for each column



`df.plot.scatter(x='w',y='h')`
Scatter chart using pairs of points



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

+

bdf

x1	x3
A	T
B	F
D	T

=

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

x1	x2
C	3

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3

+

zdf

x1	x2
B	2
C	3
D	4

=

Set-like Operations

x1	x2
B	2
C	3

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```
Rows that appear in ydf but not zdf (Setdiff).

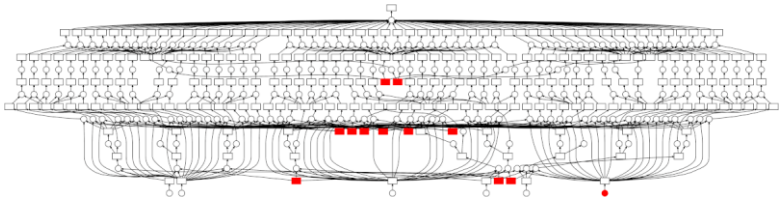
Pandas advantages (over SQL)

- Pandas Dataframes:
  - friendly support for human development and debugging process
  - No verbose and illegible nesting
- Python universe
  - Visualisation
  - Machine learning
  - …

Pandas, Dask, DBMS…

- Memory limit?? (cache management,….)
- Cost-based query optimisation?
- (ACID transactions?)
- …

- Dask DataFrames
  - Memory limit, parallel processing: OK
  - Task graph



Scalable data processing, NoSQL

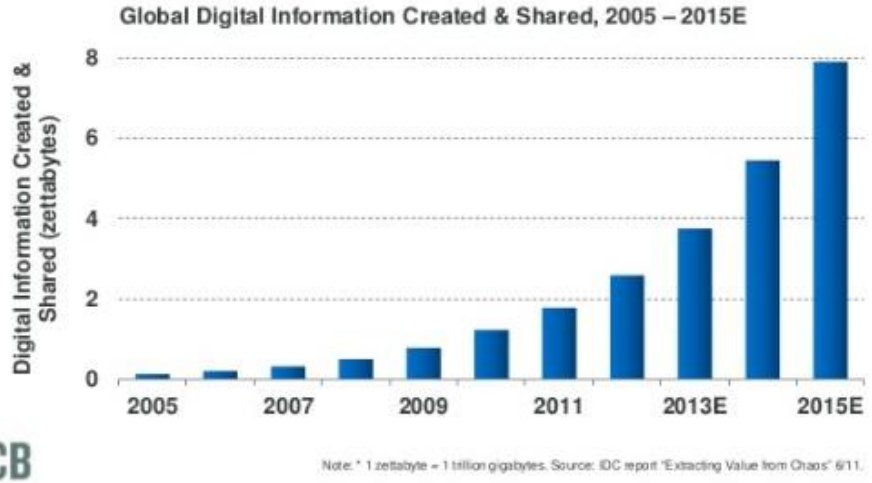
Gergely Lukács
Pázmány Péter Catholic University
Faculty of Information Technology
Budapest, Hungary
lukacs@itk.ppke.hu

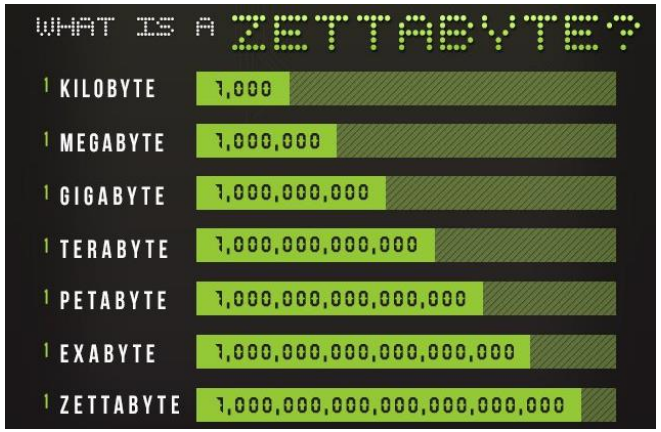
NoSQL databases

How much data?

- Google processes 20 PB a day (2008)
- Wayback Machine has 3 PB + 100 TB/month (3/2009)
- Facebook has 2.5 PB of user data + 15 TB/day (4/2009)
- eBay has 6.5 PB of user data + 50 TB/day (5/2009)
- CERN's LHC will generate 15 PB a year (??)

Amount of data doubles every 20 months





NoSQL: The Name

- "SQL" = Traditional relational DBMS
  - efficient, reliable, convenient, and safe multi-user storage of and access to massive amounts of persistent data
- Recognition over past decade or so:
  Not every data management/analysis problem is best solved using a traditional relational DBMS
  - Web-based systems!
- "NoSQL" ("No SQL")
  - Not using traditional relational DBMS
  - **Not Only SQL**

NoSQL – Definition ?

- Heterogeneous group of concepts, systems
  - Key-value stores
  - Wide column stores
  - Document stores
  - …

NoSQL – Advantages

- Depend on system/category (heterogeneous concept!!)

- Higher performance
- Easy distribution of data on nodes (sharding): scalability, fault tolerance
- Flexibility: schema free data model
- Simpler administration

NoSQL – Methods

- No normalied relational data model
- Transaction management relaxed (ACID->BASE), fewer garanties
  - **Basically available:** Nodes in the a distributed environment can go down, but the whole system shouldn't be affected.
  - **Soft State** (scalable): The state of the system and data changes over time.
  - **Eventual Consistency**: Given enough time, data will be consistent across the distributed system.
- Less powerful querying

CAP Theorem

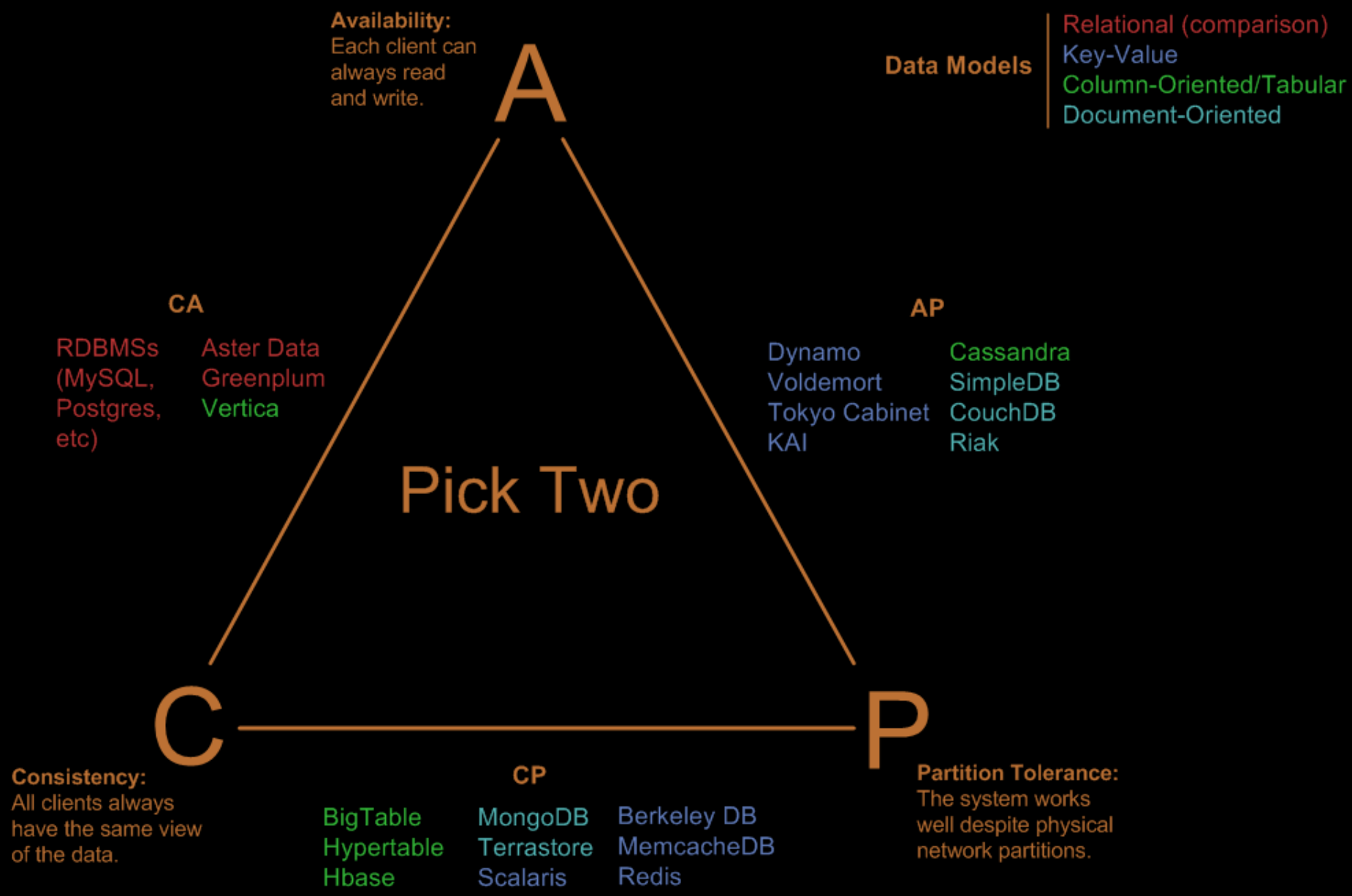
- Also known as Brewer's Theorem by Prof. Eric Brewer, published in 2000 at University of Berkeley.
- "Of three properties of a shared data system: data consistency, system availability and tolerance to network partitions, only two can be achieved **at any given moment**."
- Proven by Nancy Lynch et al. MIT labs.

- http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

CAP Semantics

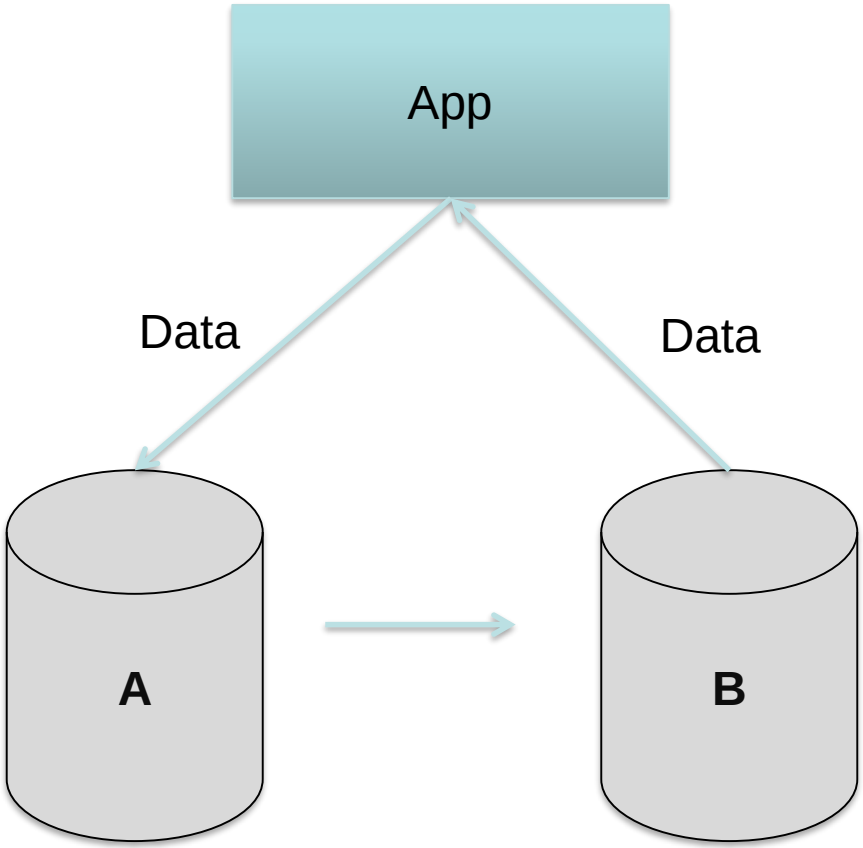
- **C**onsistency: Clients should read the same data. There are many levels of consistency.
  - Strict Consistency – RDBMS.
  - Tunable Consistency – Cassandra.
  - Eventual Consistency – Amazon Dynamo.
- **A**vailability: Data to be available.
- **P**artition Tolerance: Data to be partitioned across network segments due to network failures.

Visual Guide to NoSQL Systems



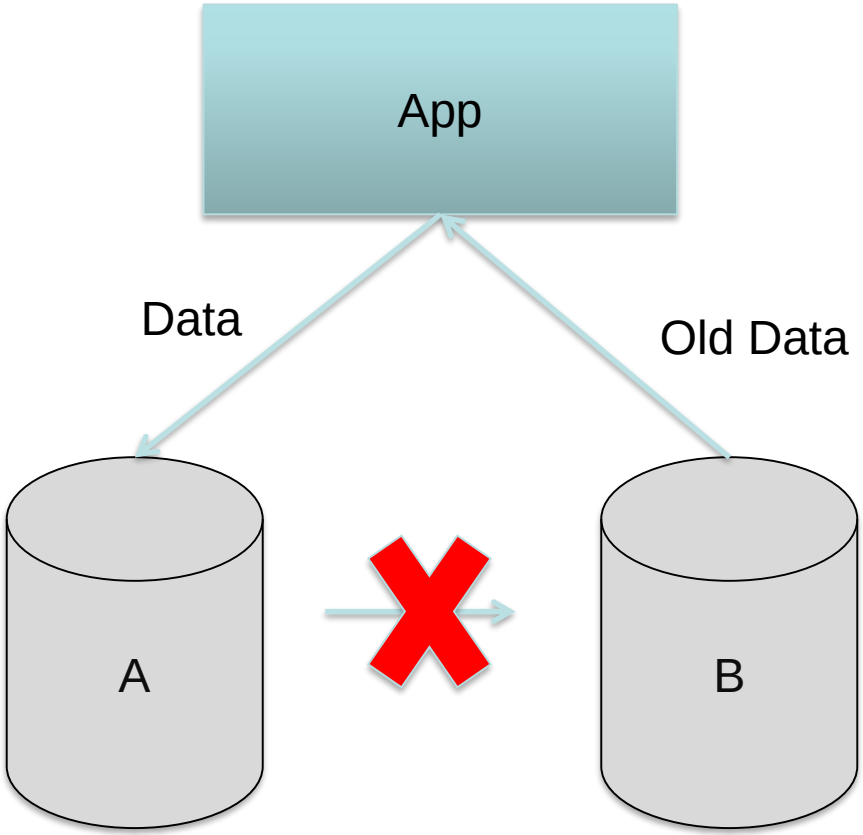
A Simple Proof

Consistent and available
No partition.



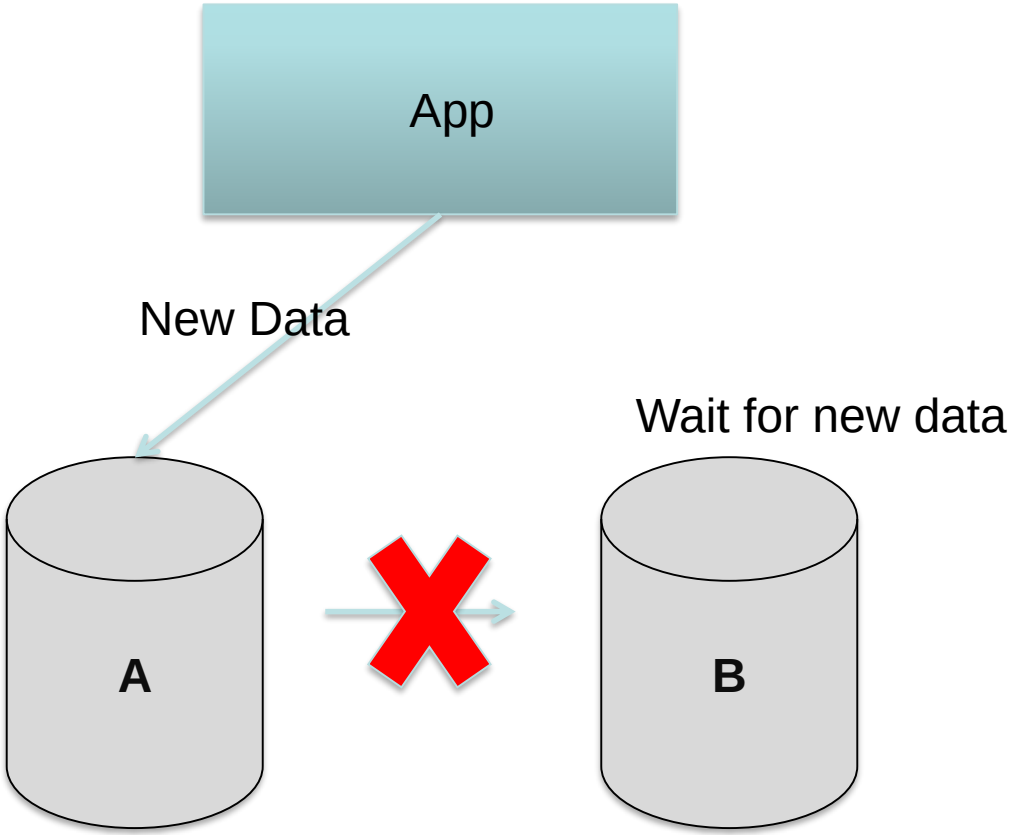
A Simple Proof

Available and partitioned
Not consistent, we get back old data.



A Simple Proof

Consistent and partitioned
Not available, waiting…



Google Cloud Spanner

Spanner is Google's highly available global SQL database [CDE+12]. It manages replicated data at great scale, both in terms of size of data and volume of transactions. It assigns globally consistent real-time timestamps to every datum written to it, and clients can do globally consistent reads across the entire

database without locking.

The CAP theorem [Bre12] says that you can only have two of the three desirable properties of:

- C: Consistency, which we can think of as serializability for this discussion;
- A: 100% availability, for both reads and updates;
- P: tolerance to network partitions.

This leads to three kinds of systems: CA, CP and AP, based on what letter you leave out. Note that you are not entitled to 2 of 3, and many systems have zero or one of the properties. For **distributed systems** over a "wide area", it is generally viewed that **partitions are inevitable**, although not necessarily common [BK14]. Once you believe that partitions are inevitable, any distributed system must be prepared to forfeit either consistency (AP) or availability (CP), which is not a choice anyone wants to make. In fact, the original point of the CAP theorem was to get designers to take this tradeoff seriously. But there are two important caveats: first, you **only** need forfeit something **during an actual partition,** and even then there are many mitigations (see the "12 years" paper [Bre12]). Second, the actual theorem is about 100% availability, while the interesting discussion here is about the **tradeoffs involved for realistic high availability**.

Key-value stores

Examples for Data

Extremely simple interface

- Data model: (key, value) pairs

Color	Red
Age	18
Size	Large
Name	Smith
Title	The Brown Dog

user1923_color	Red
user1923_age	18
user3371_color	Blue
user4344_color	Brackish
user1923_height	6' 0"
user3371_age	34

- Operations
  - Insert(key,value)
  - Fetch(key)
  - Update(key, value)
  - Delete(key)

Key-Value Stores

Implementation: efficiency, scalability, fault-tolerance

- Records distributed to nodes based on key
- Replication
- Single-record transactions, “eventual consistency”

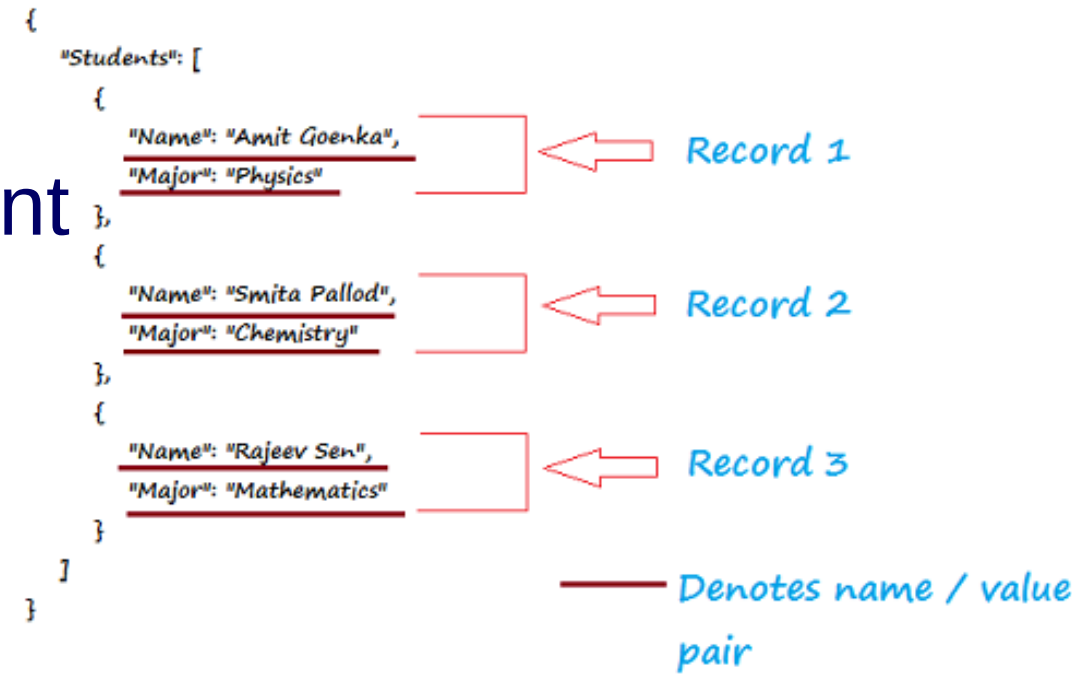
52 systems in ranking, April 2016

Rank			DBMS	Database Model	Score		
Apr 2016	Mar 2016	Apr 2015			Apr 2016	Mar 2016	Apr 2015
1.	1.	1.	Redis	Key-value store	111.24	+5.02	+16.69
2.	2.	2.	Memcached	Key-value store	28.01	-1.23	-6.04
3.	3.	3.	Amazon DynamoDB	Multi-model	23.12	+0.89	+8.54
4.	4.	4.	Riak KV	Key-value store	11.49	-0.60	-1.60
5.	5.	↑ 6.	Hazelcast	Key-value store	6.68	-0.13	+1.00
6.	6.	↓ 5.	Ehcache	Key-value store	6.46	-0.25	-1.25
7.	7.	↑ 8.	OrientDB	Multi-model	6.31	-0.35	+2.93
8.	8.	↑ 11.	Aerospike	Key-value store	4.20	+0.10	+1.71
9.	↑ 10.	9.	Oracle Coherence	Key-value store	3.13	-0.24	-0.19
10.	↓ 9.	↓ 7.	Berkeley DB	Key-value store	3.06	-0.38	-0.86
11.	11.	↓ 10.	Amazon SimpleDB	Key-value store	2.96	+0.05	-0.24
12.	12.	12.	Oracle NoSQL	Key-value store	2.51	-0.02	+0.39

Document stores

Document Stores

- Like Key-Value Stores except value is document
  - XML
  - YAML
  - JSON
  - BSON
  - binary forms
    - PDF
    - Microsoft Office documents (MS Word, Excel, and so on).



Document Stores

- Data model: (key, document) pairs
  - Basic operations:
    - Insert(key,document), Fetch(key), Update(key), Delete(key)
  - Also Fetch based on document contents

39 systems in ranking, April 2016

Rank Apr 2016	Rank Mar 2016	Rank Apr 2015	DBMS	Database Model	Score Apr 2016	Score Mar 2016	Score Apr 2015
1.	1.	1.	MongoDB	Document store	312.44	+7.11	+33.85
2.	2.	↑ 3.	Couchbase	Document store	25.02	-0.78	-0.55
3.	↑ 4.	↑ 4.	Amazon DynamoDB	Multi-model	23.12	+0.89	+8.54
4.	↓ 3.	↓ 2.	CouchDB	Document store	22.36	-1.02	-4.58
5.	5.	5.	MarkLogic	Multi-model	9.12	-0.25	-1.09

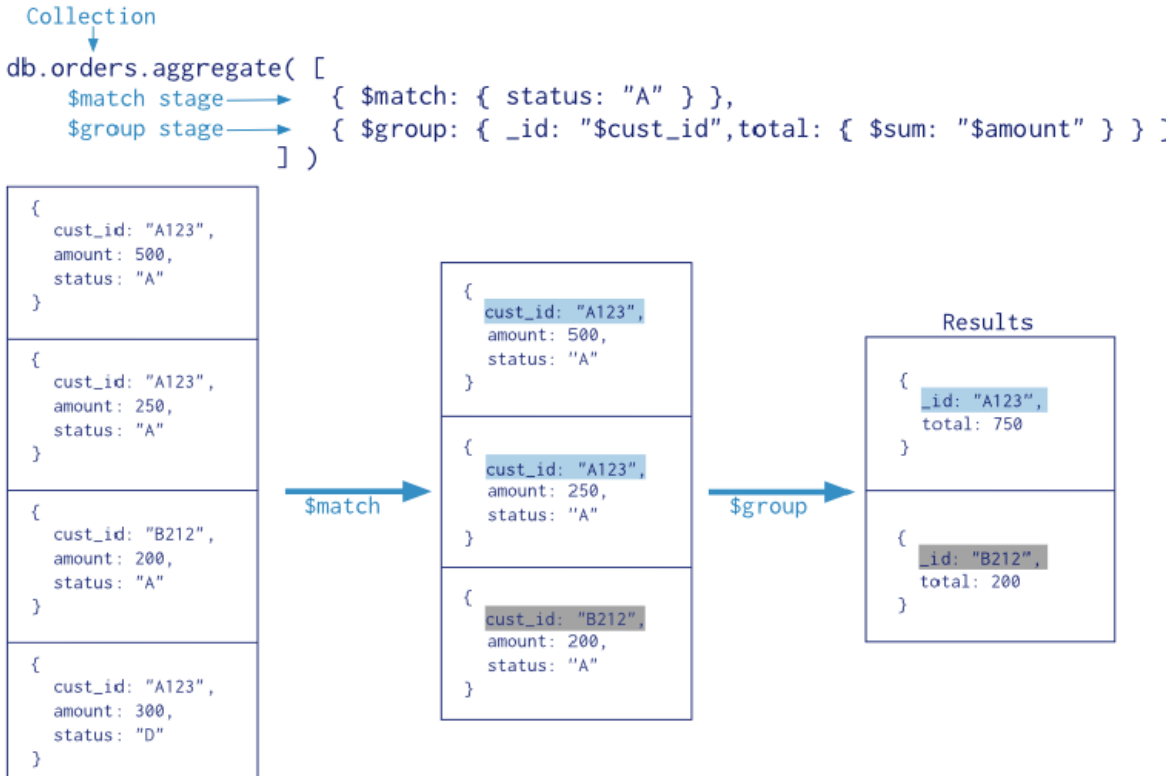
MongoDB

- High performance
- High availability
- Horizontal scalability
  - Sharding: distributing data across a cluster of machines

- Rich query language
  - Data aggregation
  - Text search
  - Geospatial queries

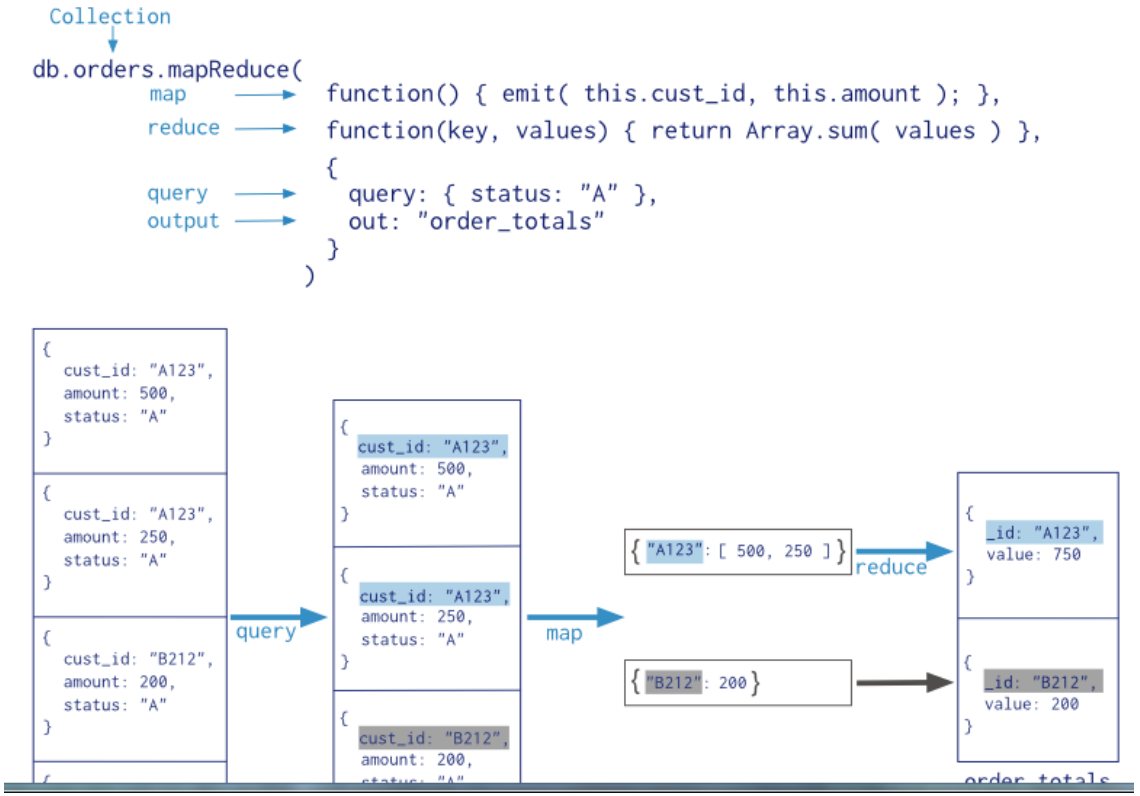
MongoDB - aggregation

- Aggregation pipeline



MongoDB - aggregation

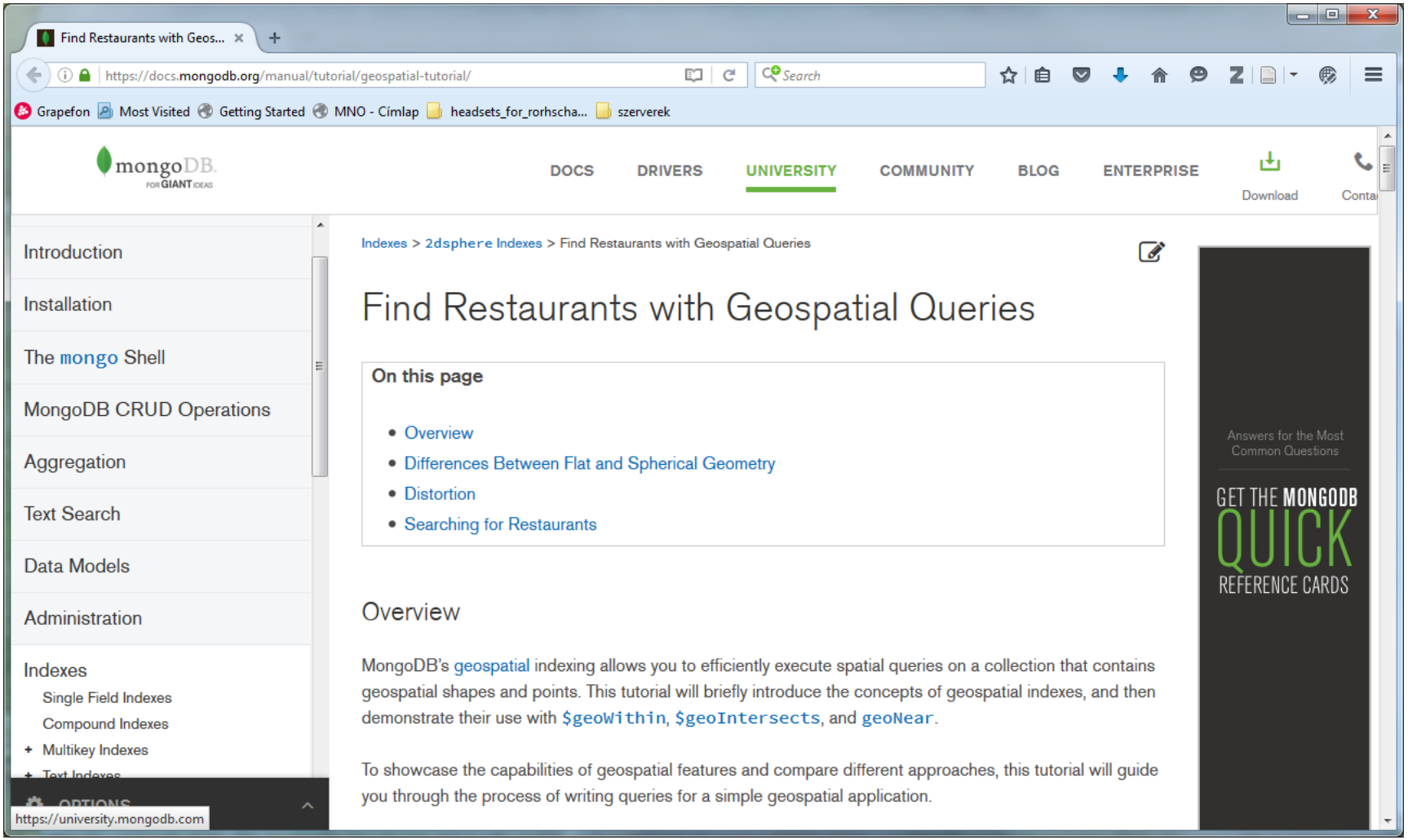
- Aggregation MapReduce



MongoDB - aggregation

- Single purpose aggregation operations
  - Db.collection.count()
  - Db.collection.group()
  - Db.collection.distinct()

MongoDB – geospatial queries



Wide column stores

- Key-value database
- Columns: name, format can vary from row to row

- Apache Cassandra