Database management, advanced Indexes

Gergely Lukács

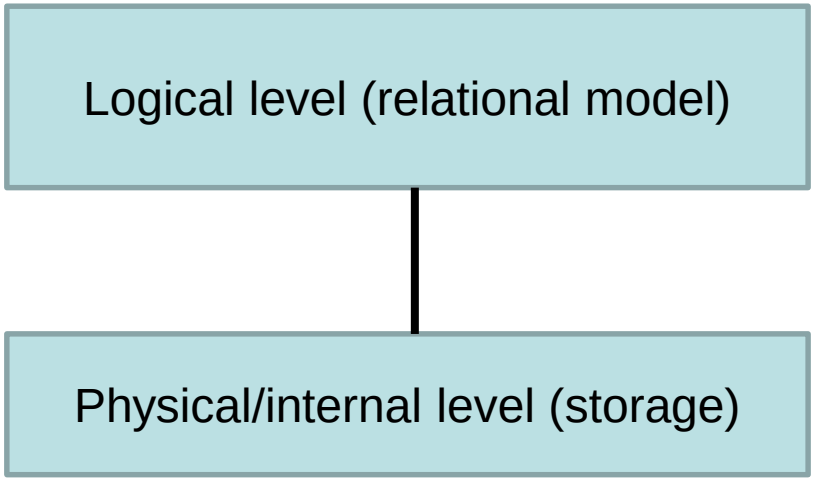
Pázmány Péter Catholic University

Faculty of Information Technology and Bionics

Budapest, Hungary

lukacs@itk.ppke.hu

DB architecture (fraction)

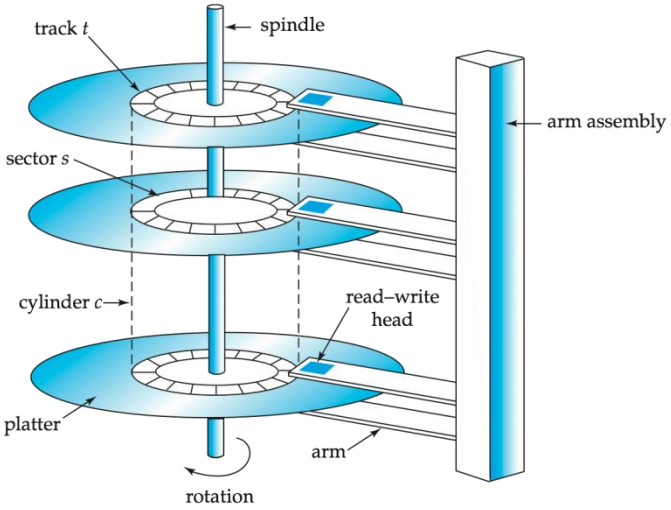


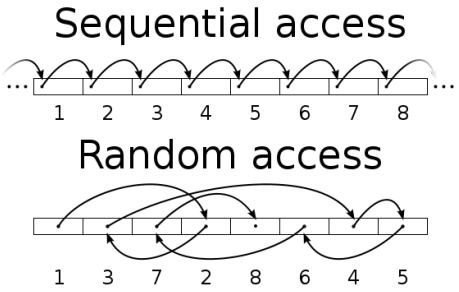
Physical data independence

Data access structures,

„Index“ (B+ tree)

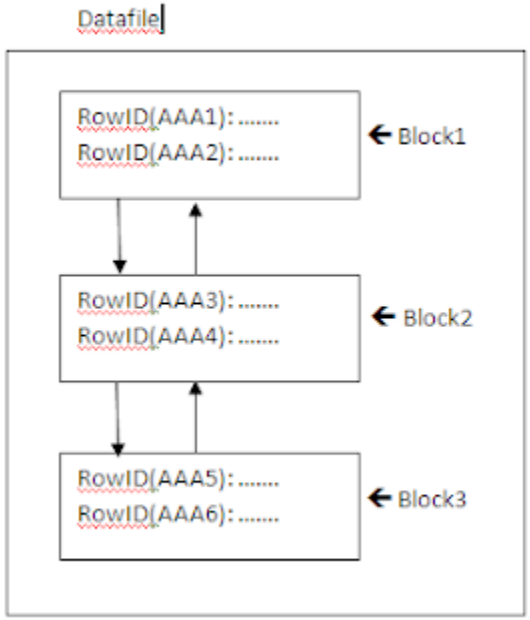
Disk, Data organisation, data access costs





- 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)

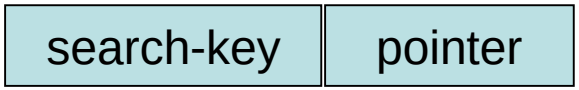
Science, technology, and mathematics [edit]

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

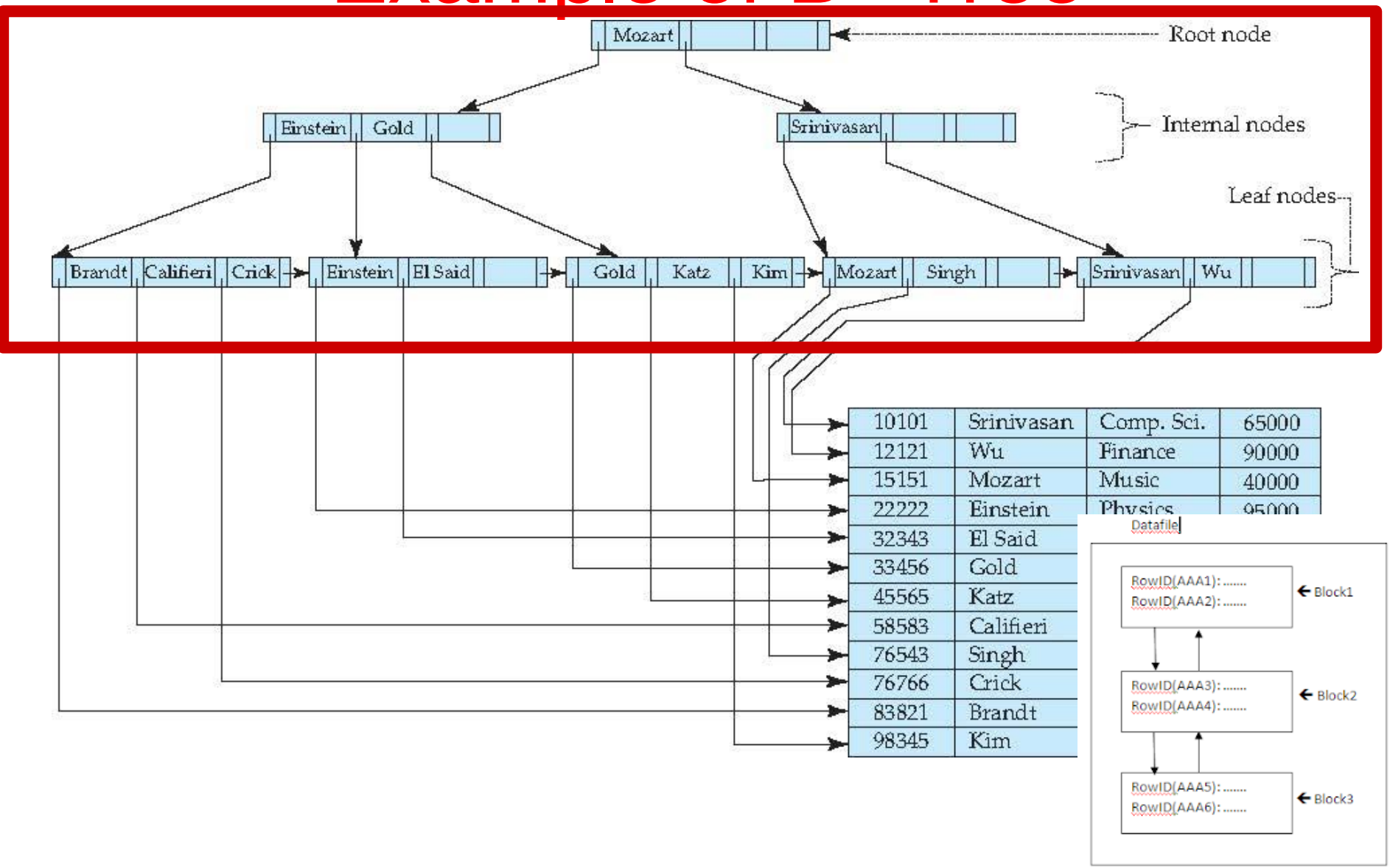
Basic Concepts

- 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

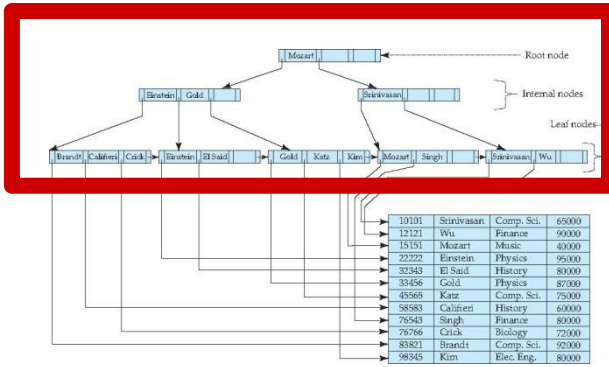


- Index files support finding an index entry for a search key efficiently (and are typically smaller than the original file)
- Two basic kinds of indices:
  - **Ordered indices:** search keys are stored in sorted order in some kind of a search tree
  - **Hash indices:** search keys are distributed uniformly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

Example of B+-Tree

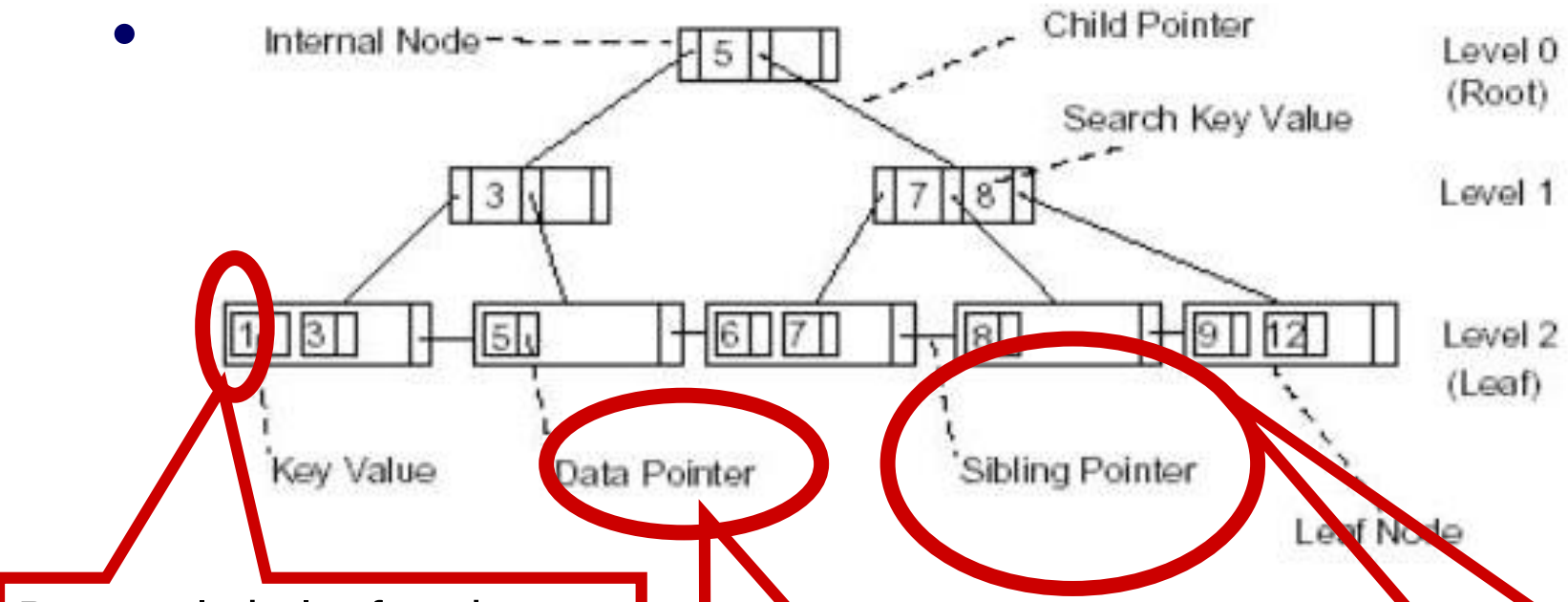


B+ tree



- 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

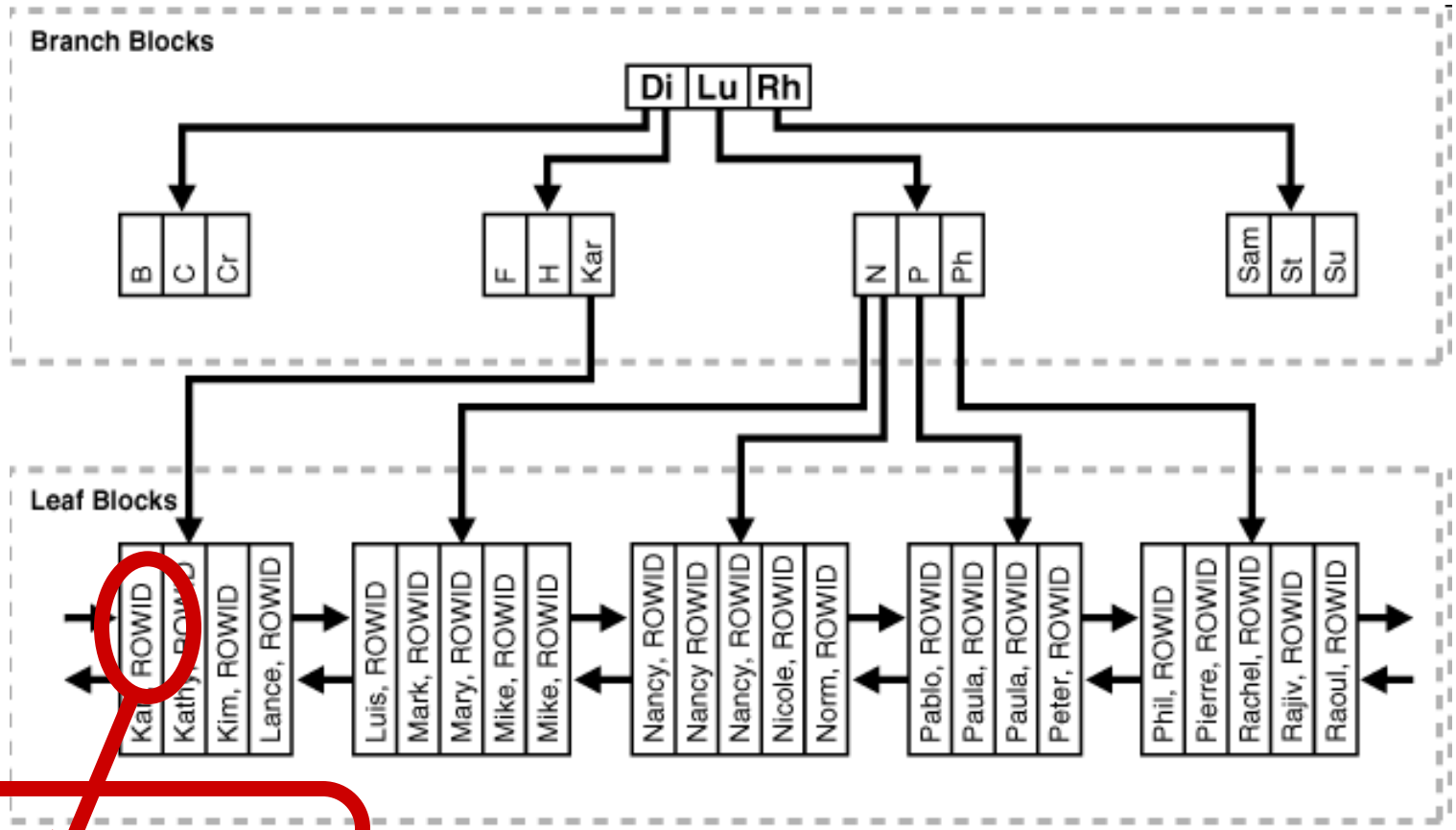


Table (record)

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.

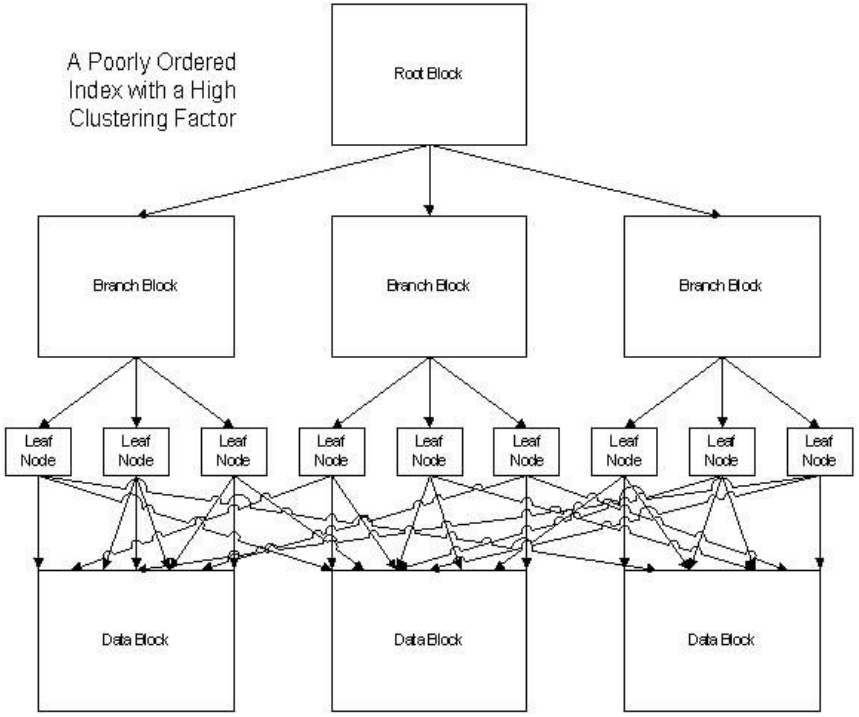
Bulk Loading and Bottom-Up Build

- 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)

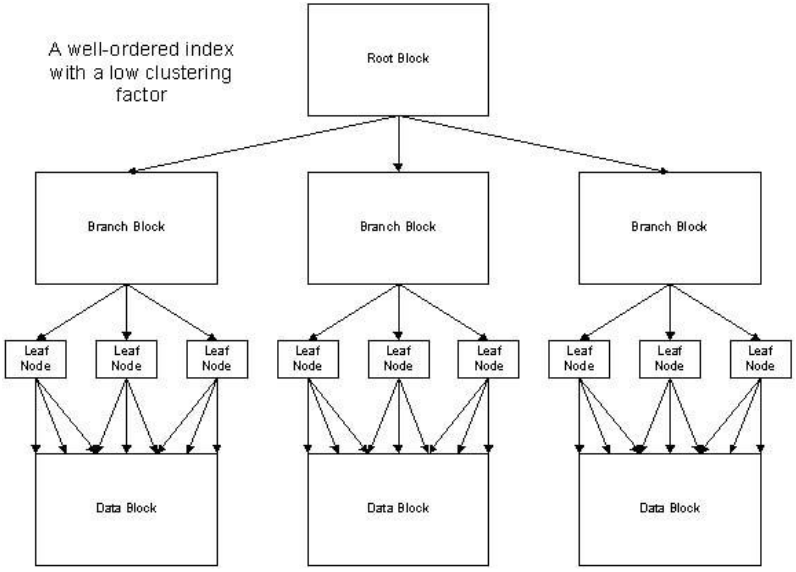
Poorly ordered table related to an index

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

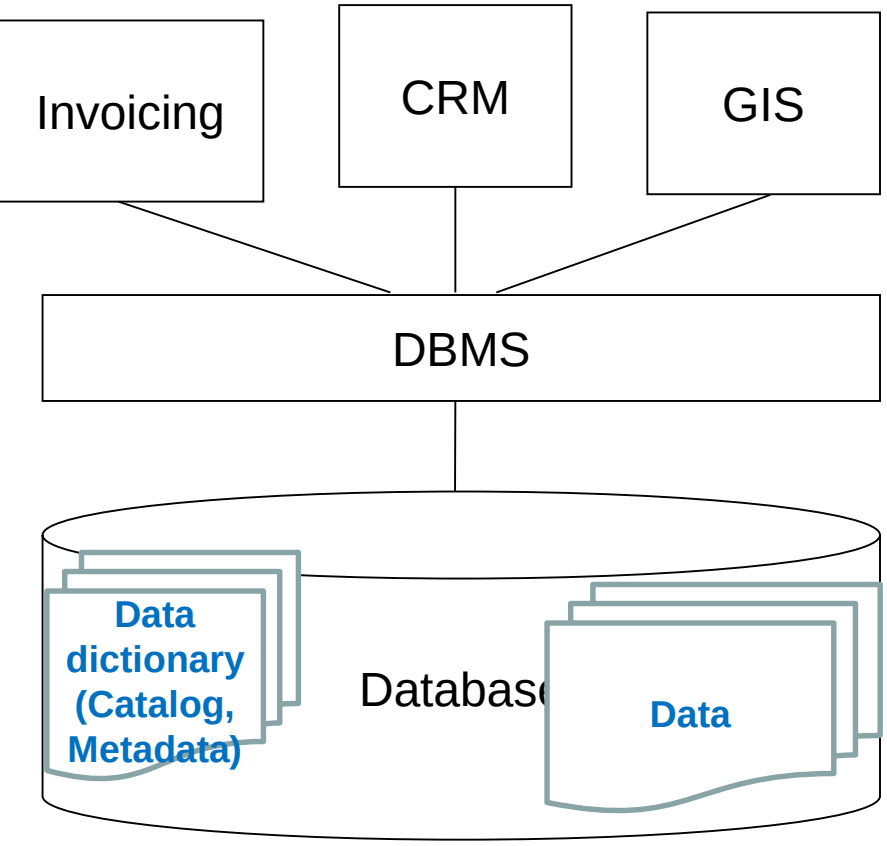


index is useful for a query condition with at most ~ ~ 5% selectivity! (rule of thumb)

Well ordered table related to an index



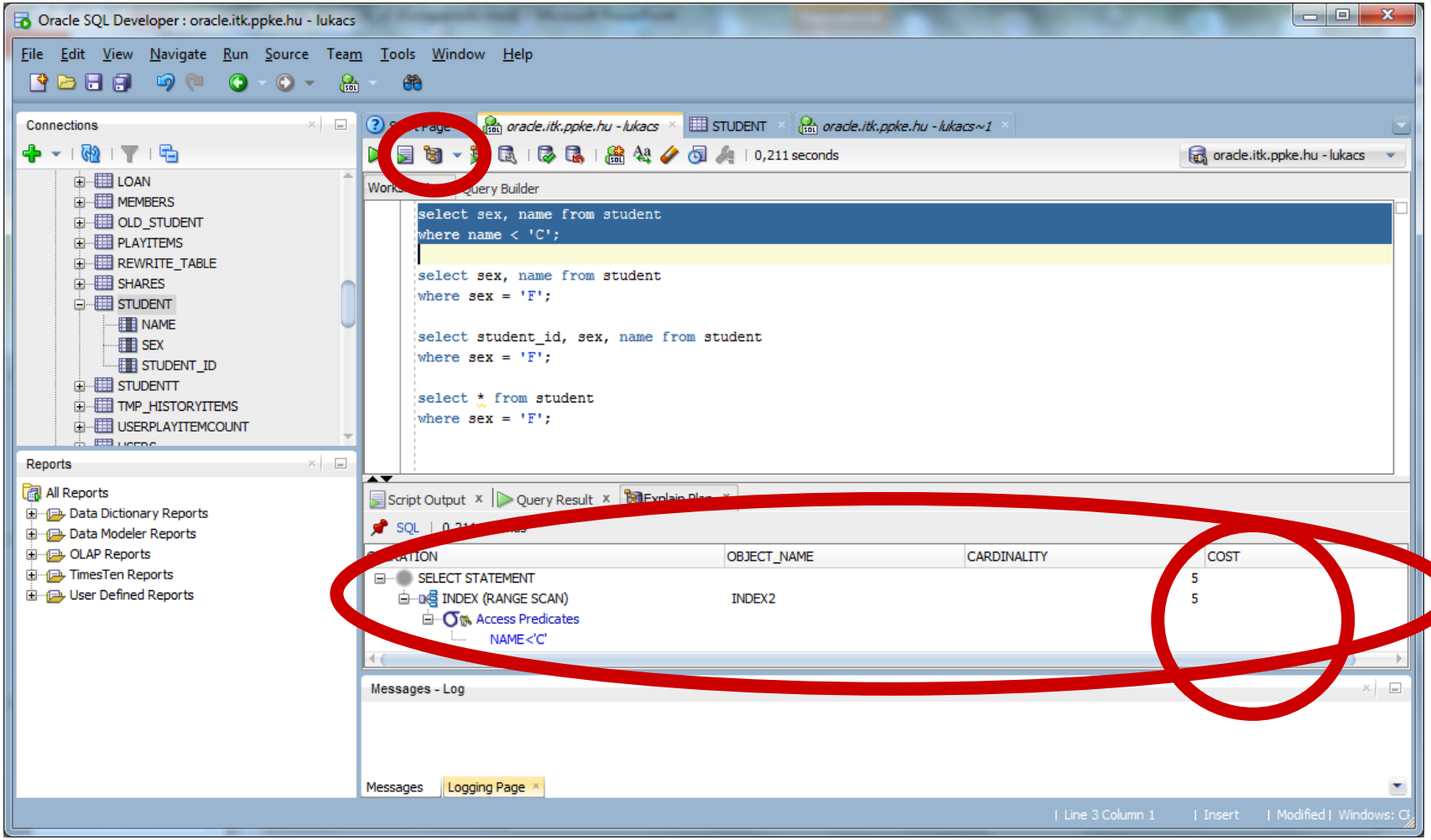
Data dictionary



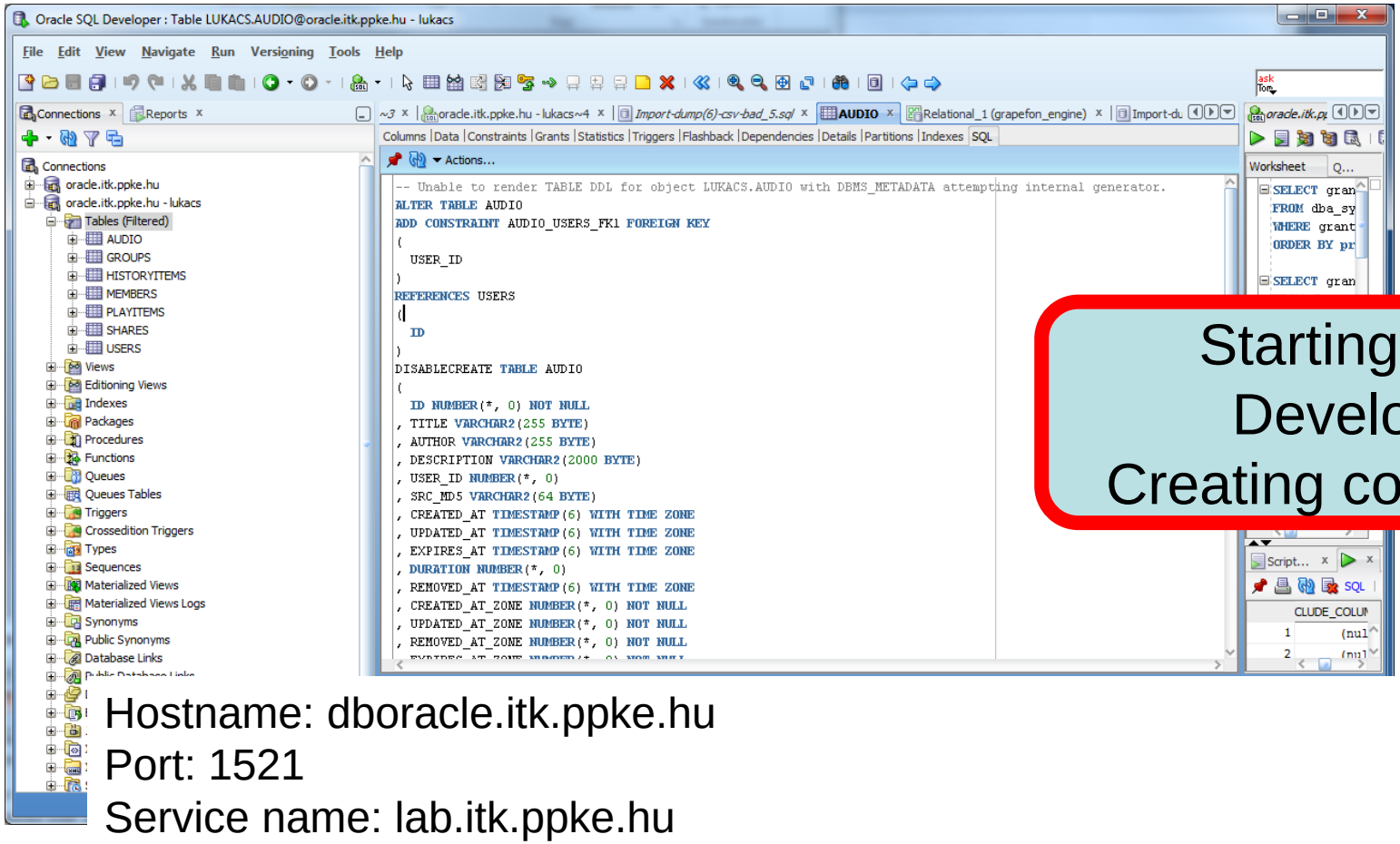
- Home / Database / Oracle Database Online Documentation 12c Release 1 (12.1) / Database Administration
- Database Reference
  - **all_tables**
  - **all_indexes (https://docs.oracle.com/database/121/REFRN/GUID-E39825BA-70AC-45D8-AF30-C7FF561373B6.htm#REFRN20088)**
  - **SQL> select value from v$parameter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