

Preferred Audio Configuration

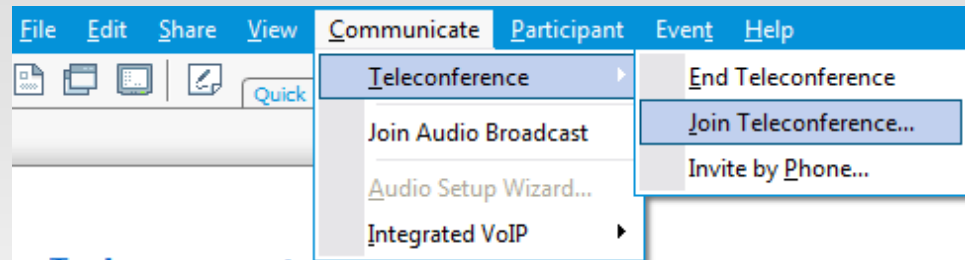
Audio stream is broadcasted to all attendees automatically. Use your **headset** or PC **speakers** to follow the webinar.

Alternative Audio Configuration

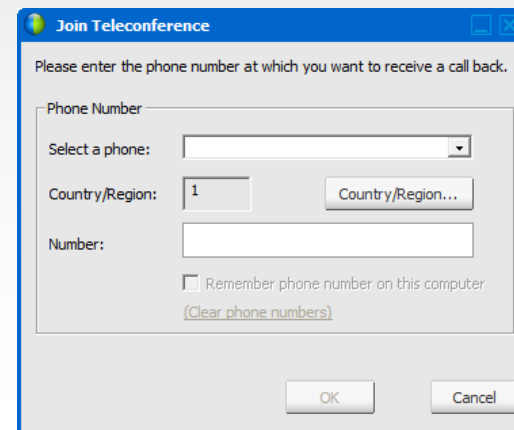
If headset or PC speakers are not available, you can join the audio conference by receiving a call back (toll !). The following steps have to be done:

1. The call back function has to be unlocked. This will be done by the host just before start of the webinar. No action is required by the attendees.

2. Join Teleconference via the menu



3. Select your country and fill in your phone number without leading Zero.

A screenshot of a dialog box titled 'Join Teleconference'. The dialog contains the following fields and controls:

- A text prompt: 'Please enter the phone number at which you want to receive a call back.'
- A 'Phone Number' section with a 'Select a phone:' dropdown menu.
- A 'Country/Region:' field with a dropdown menu showing '1' and a 'Country/Region...' button.
- A 'Number:' text input field.
- A checkbox labeled 'Remember phone number on this computer'.
- A link labeled '(Clear phone numbers)'.
- 'OK' and 'Cancel' buttons at the bottom.

SQL Workload Tuning with DB2

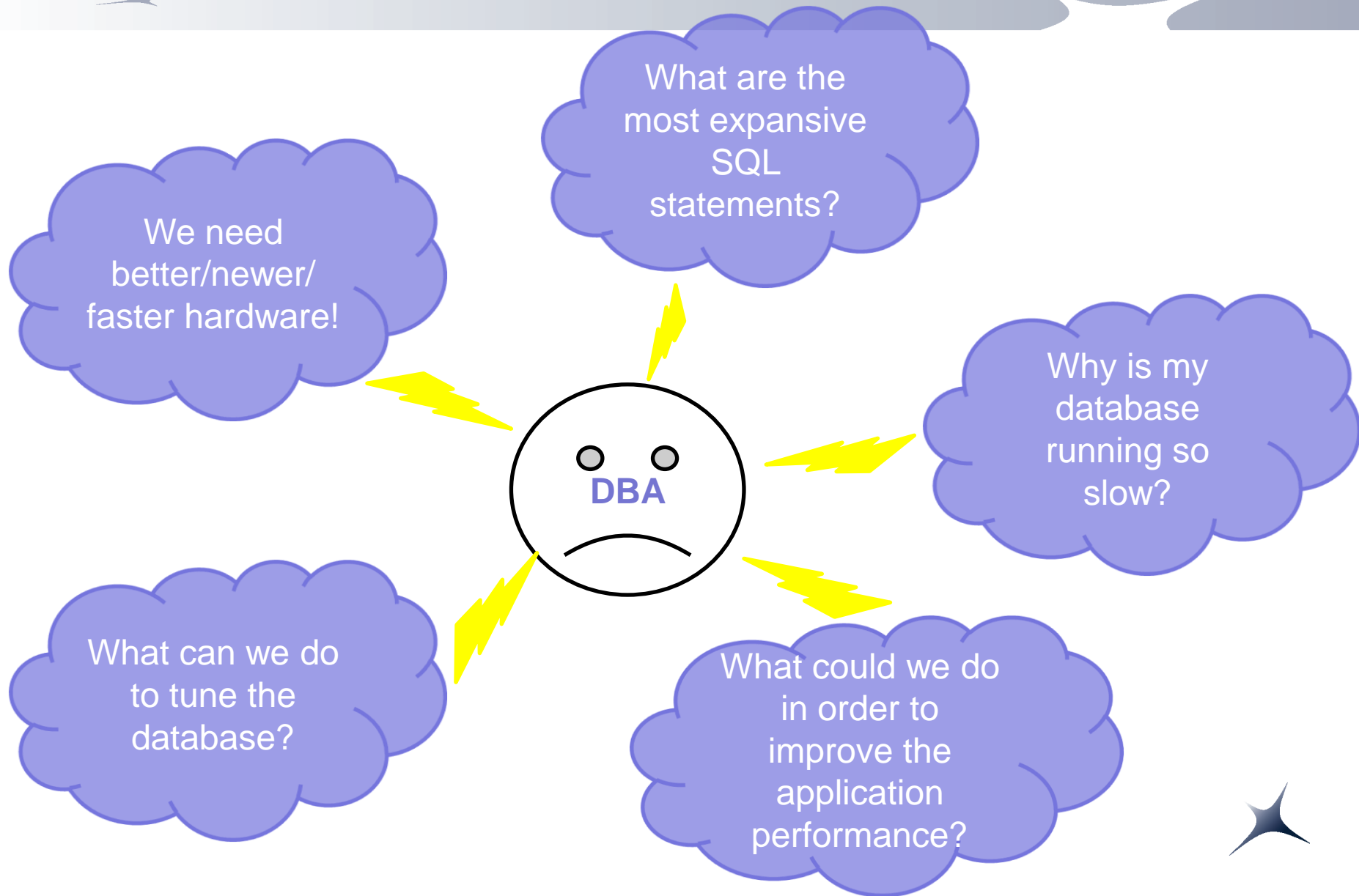
Thomas Sprenger

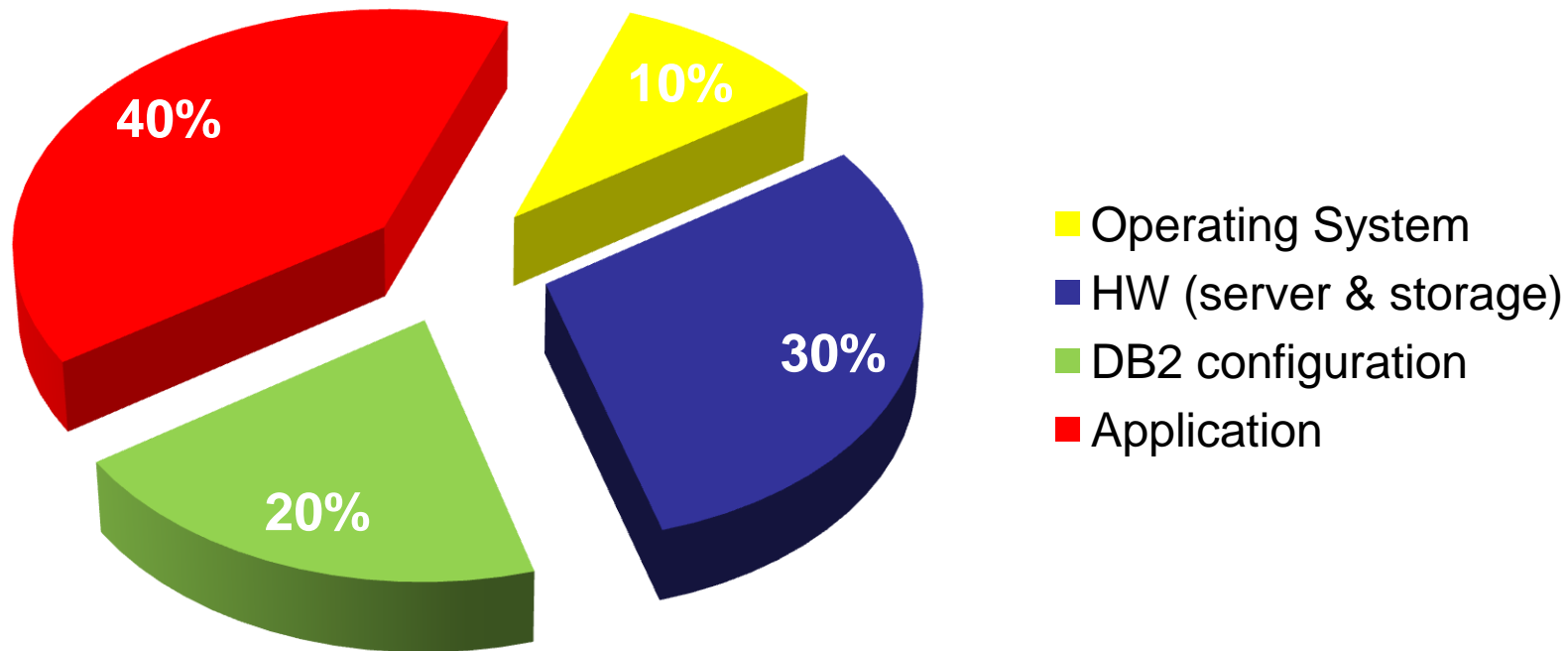
IT-Consultant

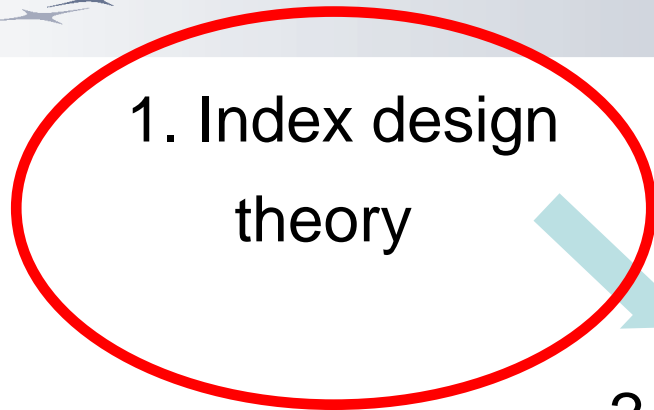
ITGAIN GmbH



Reasons for this presentation







2. Collect workload

3. Group workload

4. Analyse workload

4. Create indexes

5. Evaluate indexes



Definition:

- An index design is the sum of all indexes in a database
- A workload consists of all SQL statements executed against a database in a specific time frame

The quality of an index design can only be measured with a specific workload. (workload A not equal workload B!)

The workload is subject to constant change due to changes in applications.

There are two different approaches of index design:

- Proactive index design
- Reactive index design



The three star approach:

First star:

- All predicates of the statements will be found at the beginning of index (higher selectivity first)

Second star:

- Add columns for ordering in index → avoids sorts

Third star:

- All requested columns are part of index → index only access

Be careful: UID operations will get slower the more indexes you create!



Example:

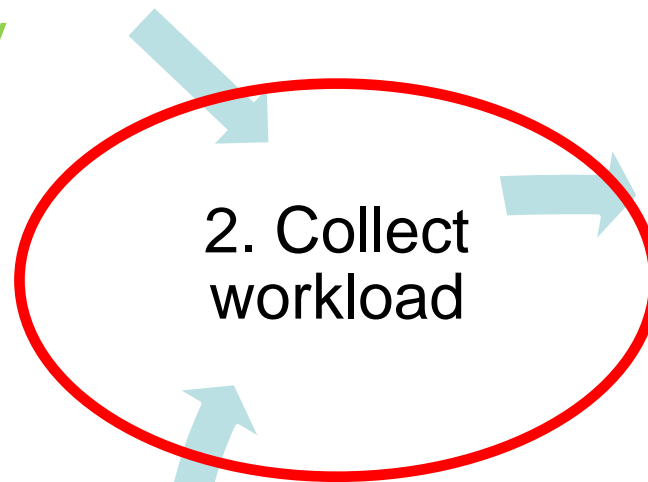
```
SELECT
  fistname,
  lastname,
  salary
FROM
  employee_sales
WHERE
  sex = 'M'
  AND salary > 65000
ORDER BY
  firstname
;
```

1. Matching predicate: sex
2. Range predicate: salary
3. Sorting: firstname
4. Rest: lastname

```
CREATE INDEX employee_sales_i1
ON employee_sales
( sex desc,
  salary,
  firstname asc,
  lastname) → optional
;
```

Remember: Keep index slices as small as possible, sort columns in index; higher selectivity always first!

1. Index design
theory



2. Collect
workload

3. Group
workload

4. Analyse
workload

4. Create
indexes

5. Evaluate
indexes



2. Collecting your workload

What are my most important SQL statements?

How to find out usually?

- phone calls from frustrated users
- bad performance metrics
- missed SLA metrics
- SQL Reports sorted by elapsed time or number of executions
- **application snapshots** taken by you
- tools (like IBM Performance Manager or Speedgain)

You can find out yourself on a regular basis → next slides!



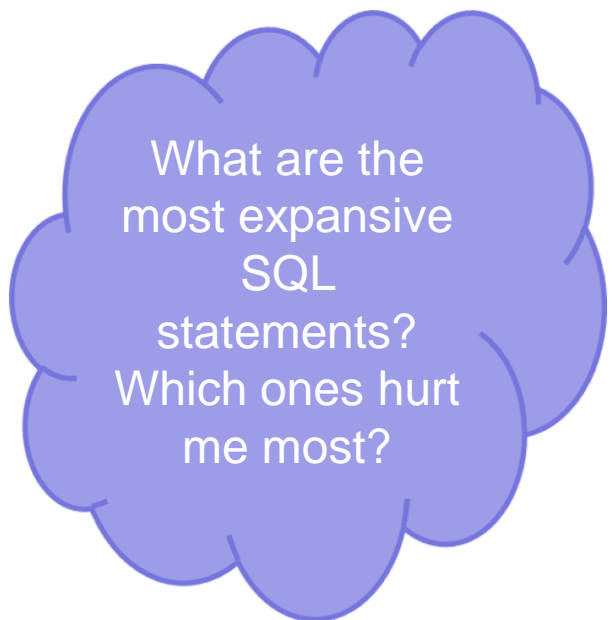
2. Collecting your workload

Where do I find the information?

- DB2 snapshots
- DB2 table functions
- DB2 administrative views

We use DB2 table function here:

`sysproc.snap_get_dyn_sql_v95`



What are the most expansive SQL statements?
Which ones hurt me most?

You could also use `sysibmadm.snapdyn_sql` administrative view or other table functions depending on your DB2 version!



Script available!

First step

Create a table to store SQL Statements and collect performance indicators ...

```
CREATE TABLE TOM.DYNSQL_COLL
(
  NUM_EXECUTIONS          BIGINT,
  TOTAL_EXEC_TIME         BIGINT,
  LAST_NUM_EXECUTIONS     BIGINT,
  LAST_TOTAL_EXEC_TIME    BIGINT,
  STMT_TEXT                VARCHAR(32000),
  LAST_USED                TIMESTAMP
)
;
```

You can always add new columns to gather different information about your workload!!



Second step

Problem:

The same SQL statement could be stored multiple times
→ Hard to analyse with “group by” clause

Solution:

Insert the SQL statement if not existing, otherwise update performance indicators

```
INSERT INTO TOM.DYNSQL_COLL
SELECT NUM_EXECUTIONS ,
       TOTAL_EXEC_TIME,
       STMT_TEXT
FROM
  TABLE(sysproc.snapshot_dyn_sql_v95( pdb35
65',-1)) AS temp
WHERE STMT_TEXT LIKE '%select%'
      OR STMT_TEXT LIKE '%SELECT%'
```

→MERGE is perfect for this job

```
MERGE INTO <tablename> USING <table reference> ON <search condition> WHEN
<matching condition> THEN <insert|update|delete operation> ELSE IGNORE
```



Using the build-in DB2 table functions to get workload:

Script available!

```
MERGE INTO TOM.DYNSQL_COLL twl
USING (SELECT num_executions,
          CAST(stmt_text AS VARCHAR(32000)) AS stmt_text,
          total_exec_time
        FROM Table(sysproc.snap_get_dyn_sql_v95('pdb3565', -1)) AS TEMP
        WHERE stmt_text LIKE '%select%'
              OR stmt_text LIKE '%SELECT%') sgds
ON ( twl.stmt_text = sgds.stmt_text )
...<continued>...
```

Pitfall ahead: Performance indicators are cumulative (e.g. CPU execution time) → if statement is existing, use „old“ performance indicator and add new values to it



2. Collecting your workload

Script available!

...<continued>...

```
WHEN MATCHED AND twl.last_num_executions < sgds.num_executions THEN
```

```
  UPDATE SET num_executions = num_executions + sgds.num_executions -  
            twl.last_num_executions,
```

```
            total_exec_time = total_exec_time + sgds.total_exec_time -  
            twl.last_total_exec_time,
```

```
            last_num_executions = sgds.num_executions,
```

```
            last_total_exec_time = sgds.total_exec_time,
```

```
            last_used=current timestamp
```

```
WHEN MATCHED AND twl.last_num_executions > sgds.num_executions THEN
```

```
  UPDATE SET num_executions = num_executions + sgds.num_executions,
```

```
            total_exec_time = total_exec_time + sgds.total_exec_time,
```

```
            last_num_executions = sgds.num_executions,
```

```
            last_total_exec_time = sgds.total_exec_time
```

```
            last_used=current timestamp
```

```
WHEN NOT MATCHED THEN
```

...<continued>...



2. Collecting your workload

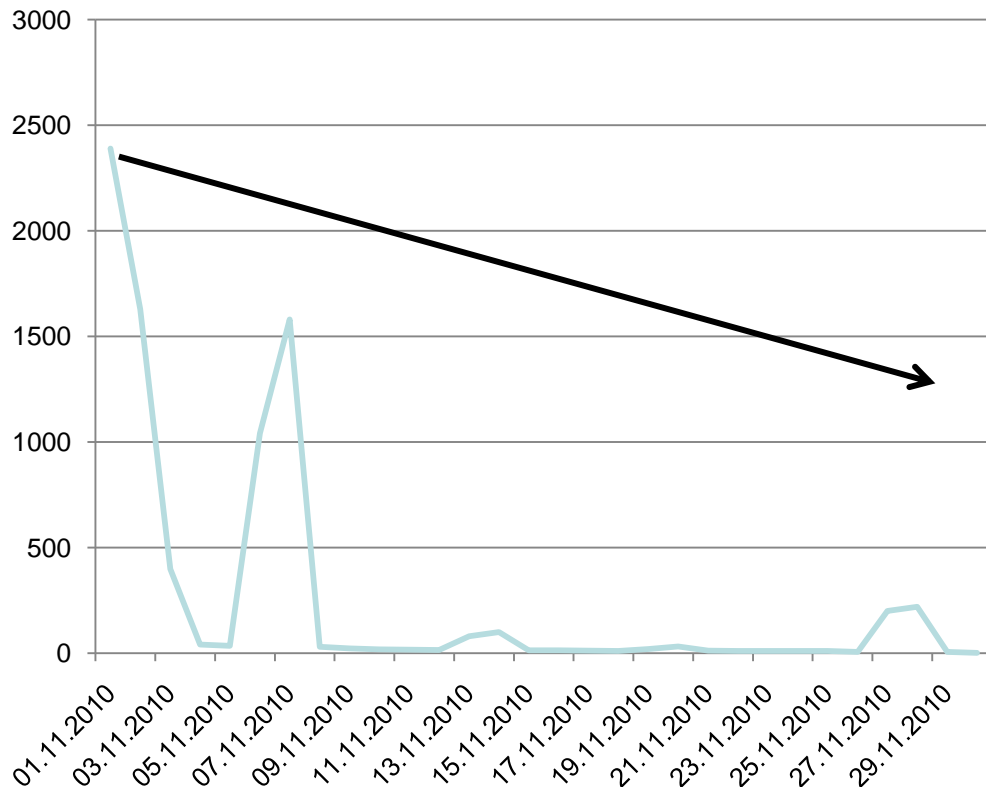
Script available!

```
...<continued>...  
INSERT(stmt_text,  
      num_executions,  
      last_num_executions,  
      total_exec_time,  
      last_total_exec_time,  
      last_used)  
VALUES(sgds.stmt_text,  
      sgds.num_executions,  
      sgds.num_executions,  
      sgds.total_exec_time,  
      sgds.total_exec_time,  
      current timestamp);
```

Finally the INSERT☺ Run this statement regularly
minutes (depending on your DB configuration)



Collecting your workload – inserts in table



Less new SQL statements collected of time. Some peaks due to weekend ETL jobs and end of months ETL jobs.

The behavior depends on your workload since:

```
Select a from b where c= 'Fritz' ;
```

is not equal to:

```
Select a from b where c= 'Otto' ;
```



1. Index design theory

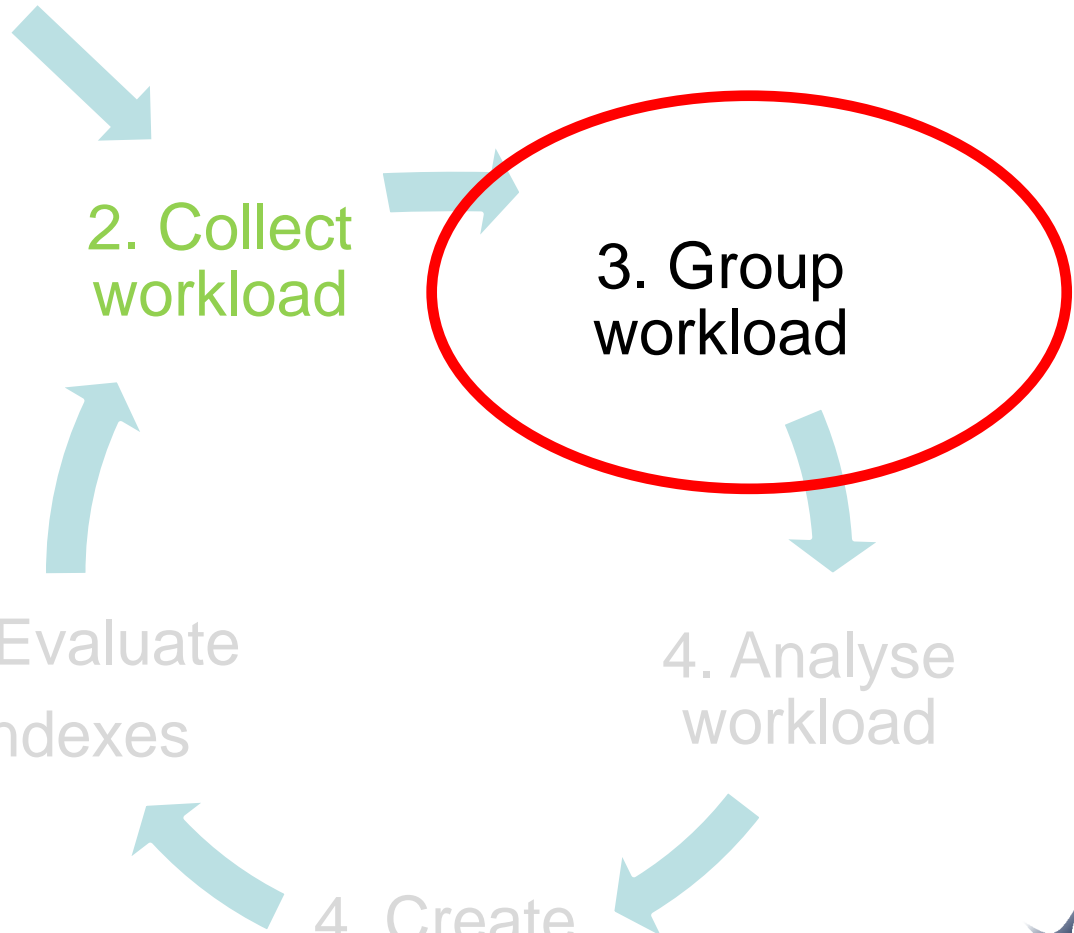
2. Collect workload

3. Group workload

4. Analyse workload

4. Create indexes

5. Evaluate indexes



Now select the statements that hurt most

→ Sorts / Sort time / CPU / IO / time

→ basically whatever you collected

→ e.g. to xx Execution time

Script available!

```
SELECT num_executions,  
       total_exec_time,  
       total_exec_time / num_executions AS time_per_exec,  
       stmt_text  
FROM   TOM.DYNSQL_COLL  
WHERE  num_executions <> 0  
ORDER BY time_per_exec desc  
FETCH FIRST 50 ROWS ONLY;
```

Remember: You got your workload of a specific time. You may also add „SET FREQUENCY“ (num_executions) to the output file!

Let's put get the most expensive statements into a file and work with it!

Script available!

```
SELECT
 '--#SET FREQUENCY ' concat
 char(a.num_executions) concat X'0A' concat
 ,a.stmt_text concat ';
FROM TOM.DYNSQL_COLL a
WHERE length(stmt_text) < 32000
   AND num_executions > 0
ORDER BY total_exec_time / num_executions DESC
FETCH FIRST 20 ROWS ONLY;
```

Extracted_
workload.
out



1. Index design
theory



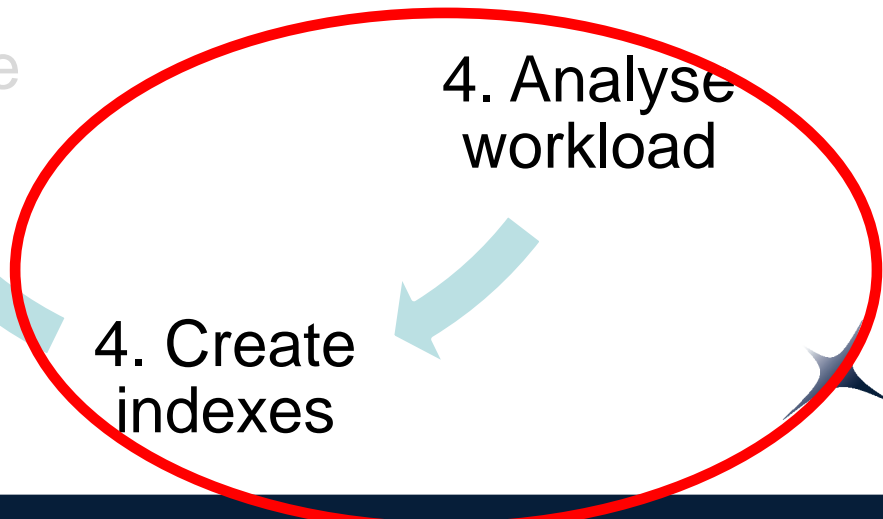
2. Collect
workload



3. Group
workload



4. Analyse
workload



4. Create
indexes



5. Evaluate
indexes



Three or more ways to do that :

1. DB2 Desing Advisor:

Pros: → Easy to use! MQTs etc. Handy tool . . .

Cons: → Result and performance advantage displayed in „DB2 dollars (timerons)“
→ Not clear which of the recommended indexes is really good

2. DB2 „set current explain mode“

Pros: → More performance indicators (IO, CPU, Total)

Cons: → Time/resource consuming, workload based results

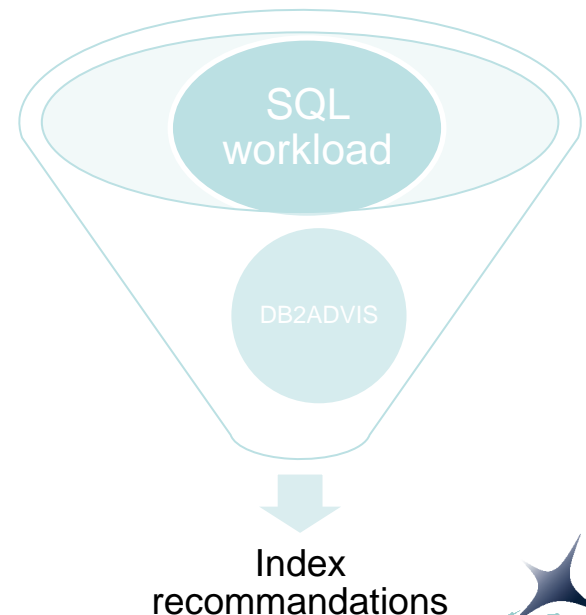
3. DB2 „enhanced current explain mode“



Run db2advis against workload:

```
# db2advis -d <database> -i advis_workload.in > advis_indexes.out
```

```
....  
Optimization finished.  
  2 indexes in current solution  
[1127,0000] timerons (without recommendations)  
[960,0000] timerons (with current solution)  
[14,82%] improvement with 1 call and no thinking 😊  
....
```



Remember: Only virtual costs „timerons“!
No qualification of indexes!



1. Explain statement

```
# db2 set current explain mode recommend indexes  
# db2 „SELECT * FROM SYSCAT.TABLES“  
# db2 set current explain mode off
```

2. Evaluate indexes

```
# db2 set current explain mode evaluate indexes  
# db2 „SELECT * FROM SYSCAT.TABLES“  
# db2 set current explain mode off
```

3. Get results

```
# db2 set current explain mode evaluate indexes  
# db2 „SELECT * FROM SYSCAT.TABLES“  
# db2 set current explain mode off
```





4.2 DB2 „set current explain mode“

The principal behind it:

„Recommend indexes and use one at a time to find the one that is best for your workload!“

Script available!
Advanced_advis
e.pl (PERL
Windows; Unix)

Index	Use
IDX101071453230000	N
IDX101071453480000	N
IDX101071454130000	N
IDX101071454440000	N
IDX101071454500000	N
IDX101071455340000	N
IDX101071457280000	N
IDX101071457310000	N
...	
IDX101071501210000	N

Frist run
0 % improvement
get baseline

Index	Use
IDX101071453230000	Y
IDX101071453480000	N
IDX101071454130000	N
IDX101071454440000	N
IDX101071454500000	N
IDX101071455340000	N
IDX101071457280000	N
IDX101071457310000	N
...	
IDX101071501210000	N

Second run
3.93475 % improvement

Index	Use
IDX101071453230000	N
IDX101071453480000	N
IDX101071454130000	N
IDX101071454440000	N
IDX101071454500000	N
IDX101071455340000	N
IDX101071457280000	N
IDX101071457310000	N
...	
IDX101071501210000	Y

x -run
41.93475 % impro



1. Index design
theory

2. Collect
workload

3. Group
workload

4. Analyse
workload

4. Create
indexes

5. Evaluate
indexes



DB2 version < v9.7:

DB2PD (Problem Determination ☺) helps you!

```
# db2pd -db pdb3565 -tcbstats index
```

```
....
```

```
TCP Index Stats
```

```
Address  TableName  IID  Scans
```

```
xxxx    yyyyy    3    0
```

```
zzzzz   aaaaa    1    23
```

```
....
```

UNIX: awk and grep will help to format output of db2pd!

The good news



DB2 version v9.7:

Get index usage regularly and store it in table for later use:

```
SELECT
    IND.TABSCHEMA,
    IND.TABNAME,
    IND.INDNAME,
    IND.IID,
    ITF.INDEX_SCANS
FROM   TABLE(MON_GET_INDEX("", "", -1)) as ITF,
       SYSCAT.INDEXES AS IND
WHERE  ITF.TABSCHEMA = IND.TABSCHEMA
       AND ITF.TABNAME = IND.TABNAME
       AND ITF.IID = IND.IID
;
```

<http://www.ibm.com/developerworks/data/library/techarticle/dm-0910db2unusedindex/index.html>



1. Index design
theory

2. Collect
workload

3. Group
workload

4. Analyse
workload

4. Create
indexes

5. Evaluate
indexes

Questions?

→ Q&A chat

→ eMail: thomas.sprenger@itgain.de

Feedback!

→ Feedback form or mail

Scripts/presentation:

→ www.itgain.de (next few days)

Next topic ??





- 1. Index design – basics**
- 2. Collecting my SQL workload**
- 3. Tuning with DB2 Design Advisor**
- 4. Tuning with use of virtuell indexes**
- 5. Index analaysis**

