CERN School of Computing

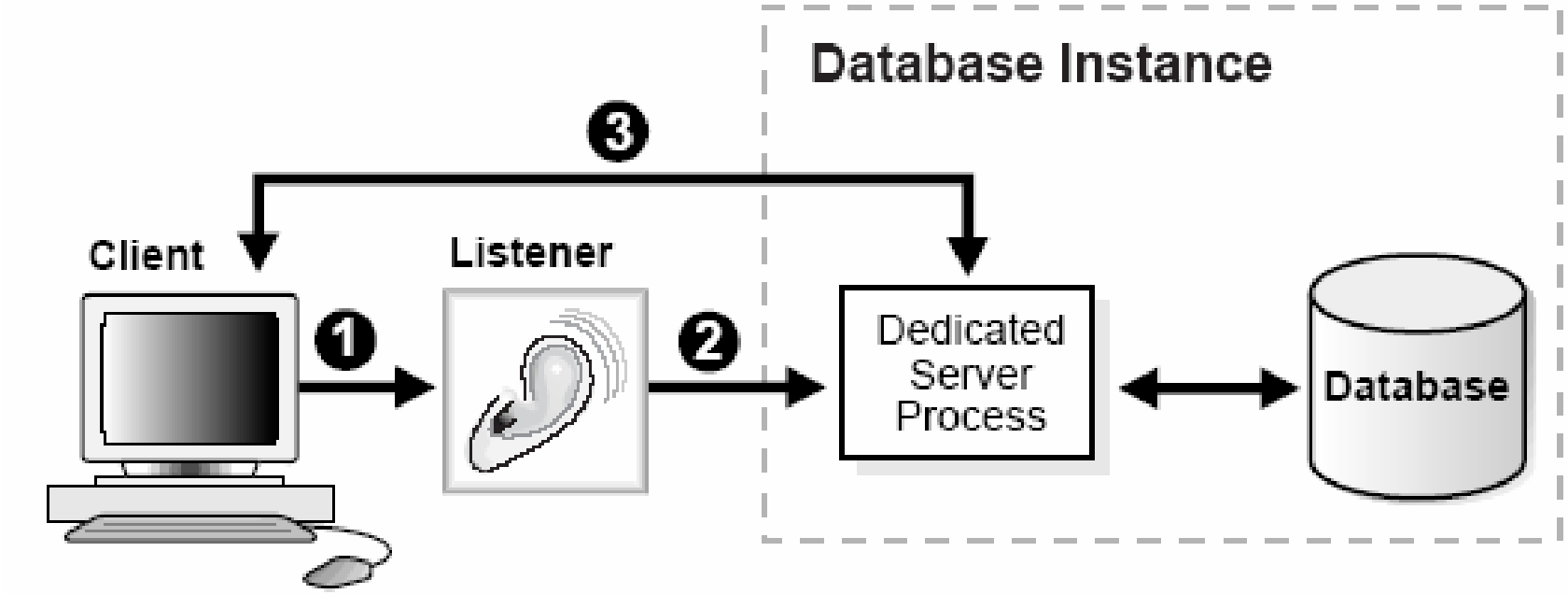
Performance Optimization and Tuning

Avoid common pitfalls (lecture plan):

- Use connection pooling
- Let the optimizer do its job
- Use bind variables
- Use appropriate tools
- Design to perform
- Don’t be too generic
- Test before going into production

CERN School of Computing

What happens when you connect to a database?



1. The listener receives a client connection request.
2. The listener starts a dedicated server process, and the dedicated server inherits the connection request from the listener.
3. The client is now connected directly to the dedicated server*).

*) This explains dedicated server process configuration, which is used more often. However, Oracle can be configured also in shared server mode.

CERN School of Computing

It happens that you process a query every time a web page is displayed

```
Connection conn = null;
Statement stmt = null;
ResultSet rset = null;
try {
    //Loading oracle jdbc driver
    Class.forName("oracle.jdbc.driver.OracleDriver");
    //Creating connection
    conn = DriverManager.getConnection(url, user,
   password);
    //Creating statement
    stmt = conn.createStatement();
    //Creating statement
    rset = stmt.executeQuery(query);

    //... processing query results ...

} catch(SQLException e) {
    //... handle exceptions ...
} finally {
    //clean up (closing resultset, statement and
   connection
    try { rset.close(); } catch(Exception e) { }
    try { stmt.close(); } catch(Exception e) { }
    try { conn.close(); } catch(Exception e) { }
}
```

You don’t want to open a new database connection every time...

CERN School of Computing

Use connection pooling

```
Connection conn = null;
Statement stmt = null;
ResultSet rset = null;
try {
    //Getting connection
    //from the pool
    conn = DBCPExample.
        getPooledConnection();
    //Creating statement
    stmt = conn.createStatement();
    //Creating statement
    rset = stmt.executeQuery(query);

    //... processing query results ...

} catch(SQLException e) {
    //... handle exceptions ...
} finally {
    /* clean up (closing resultset,
     statement and connection) */
    try { rset.close(); }
        catch(Exception e) { }
    try { stmt.close(); }
        catch(Exception e) { }
    try { conn.close(); }
        catch(Exception e) { }
}
```

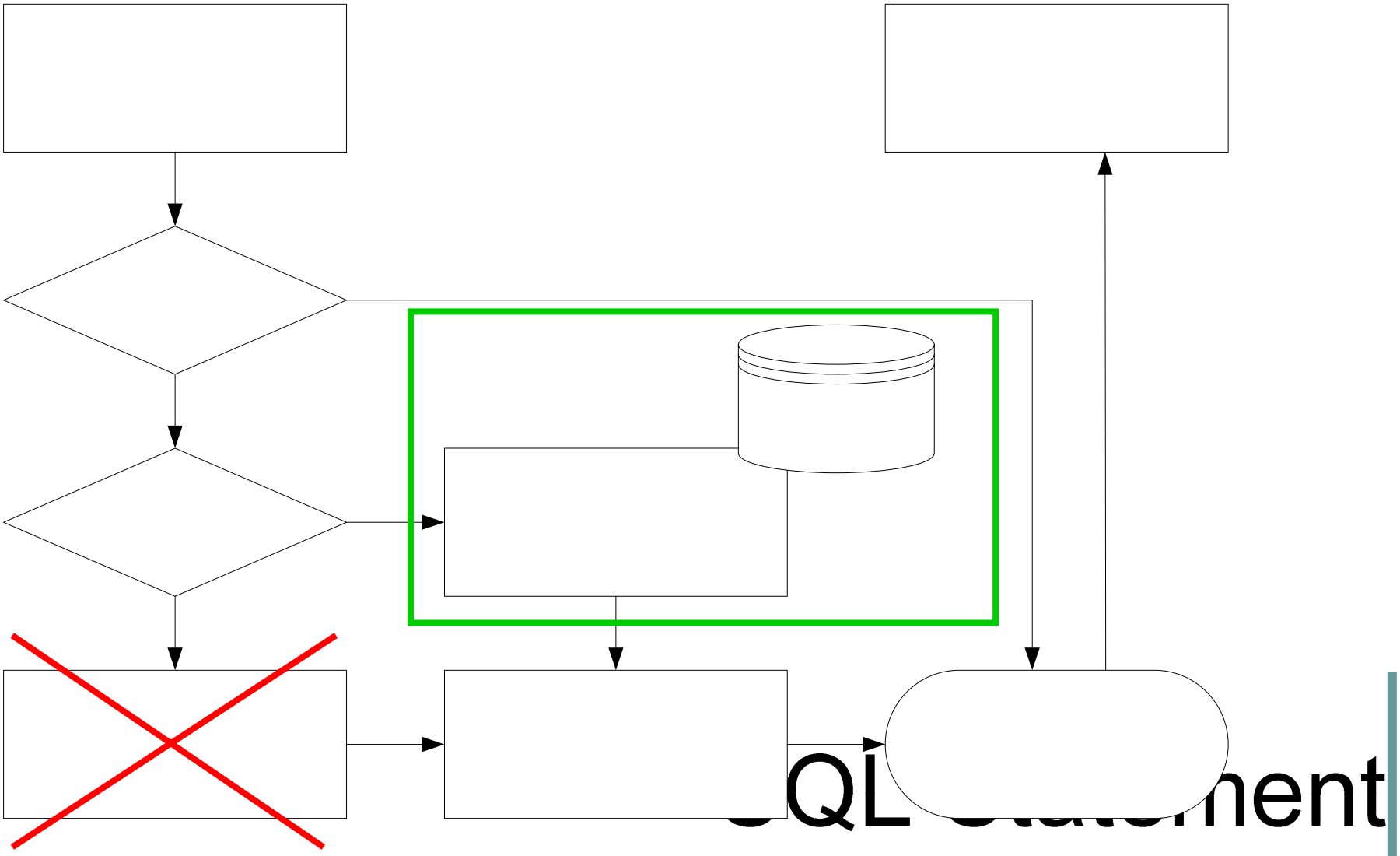
```
public static Connection getPooledConnection()
throws SQLException {
        return poolingDataSource.getConnection();
    }
```

```
private static BasicDataSource poolingDataSource = null;

public static synchronized void
initializePoolingDataSource(String url, String user, String
password) throws SQLException {

        //create new data source at set its attributes
        poolingDataSource = new BasicDataSource();

ds.setDriverClassName("oracle.jdbc.driver.OracleDriver");
        ds.setUsername(user);
        ds.setPassword(password);
        ds.setUrl(url);

poolingDataSource = ds;
    }
```

Not closing really, only returning to the pool

There is no need for connection pooling in single-user environments. But in a web application – it's a must.

What happens when you select * from emp?



CERN School of Computing

Rule Based Optimizer versus Cost Based Optimizer

- Rule Based Optimizer
  - query plans are generated according to a predefined set of rules
  - does not undestand bitmap index, function based index, partition tables...
  - disappears in Oracle 10g
- Cost Based Optimizer
  - Plans are generated based on statistics and costs associated with performing specific operations

Let the optimizer do its job!

```
BEGIN
DBMS_STATS.GATHER_SCHEMA_STATS(
   ownname=>null,
   estimate_percent=>DBMS_STATS.AUTO_SAMPLE_SIZE,
   method_opt=>'FOR ALL COLUMNS SIZE AUTO',
   cascade=>TRUE
);
END;
```

Gather statistics for all objects in a schema

Schema to analyze (null means current schema)

Let Oracle determine the best sample size for good statistics

Oracle collects histograms for all columns and determines the number of histogram buckets based on data distribution and the workload of the columns

Gather statistics on the indexes too

Stale statistics are the most common reason why the optimizer fails.

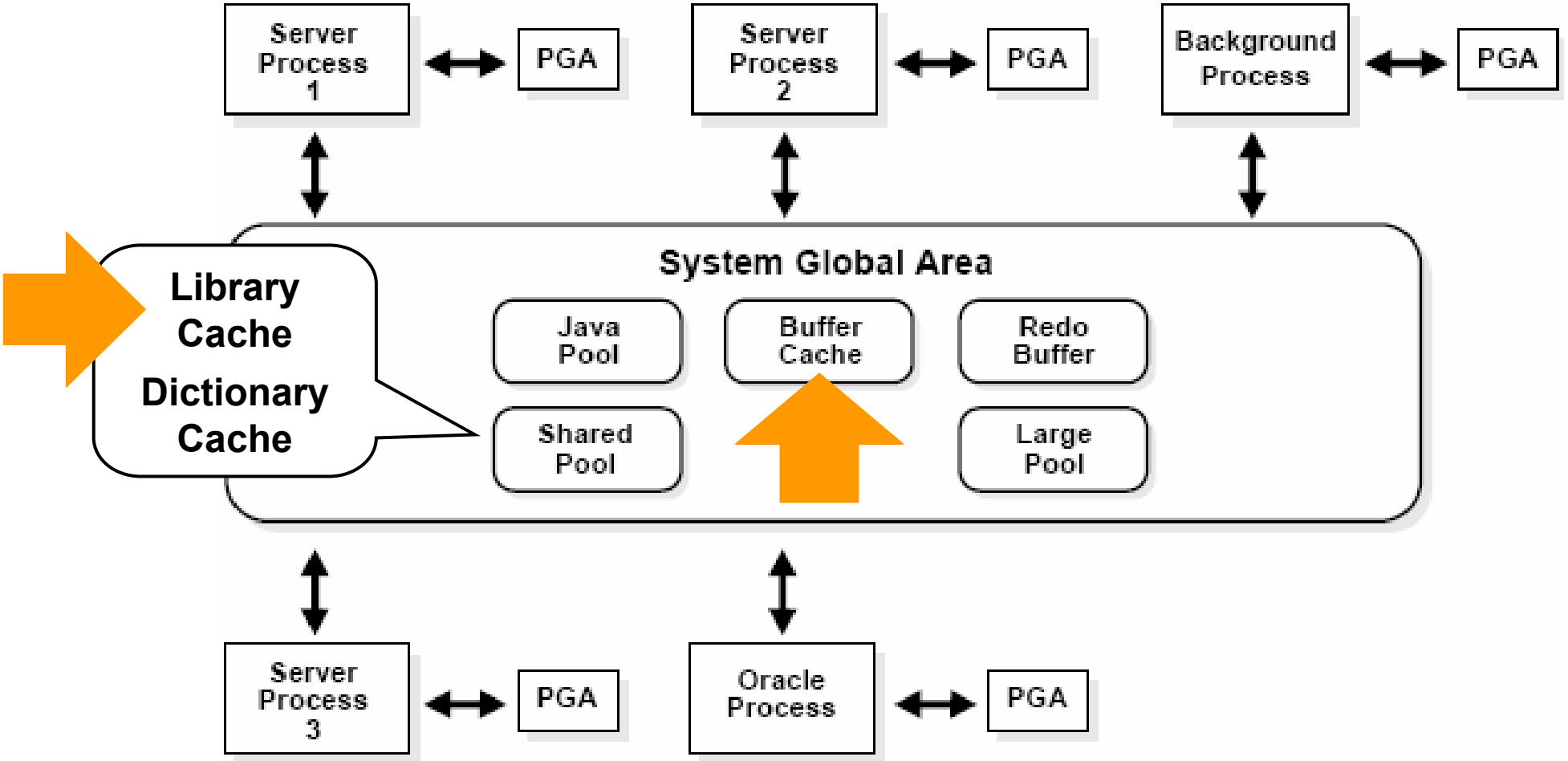
Careful with hints!

- Hints are instructions passed to the optimizer to favour one query plan over another.
- Declared with `/*+ hint hint hint … hint */`

```
select /*+ USE_INDEX(emp.ind_deptno)*/
count(*)
from emp
where deptno = 50
```

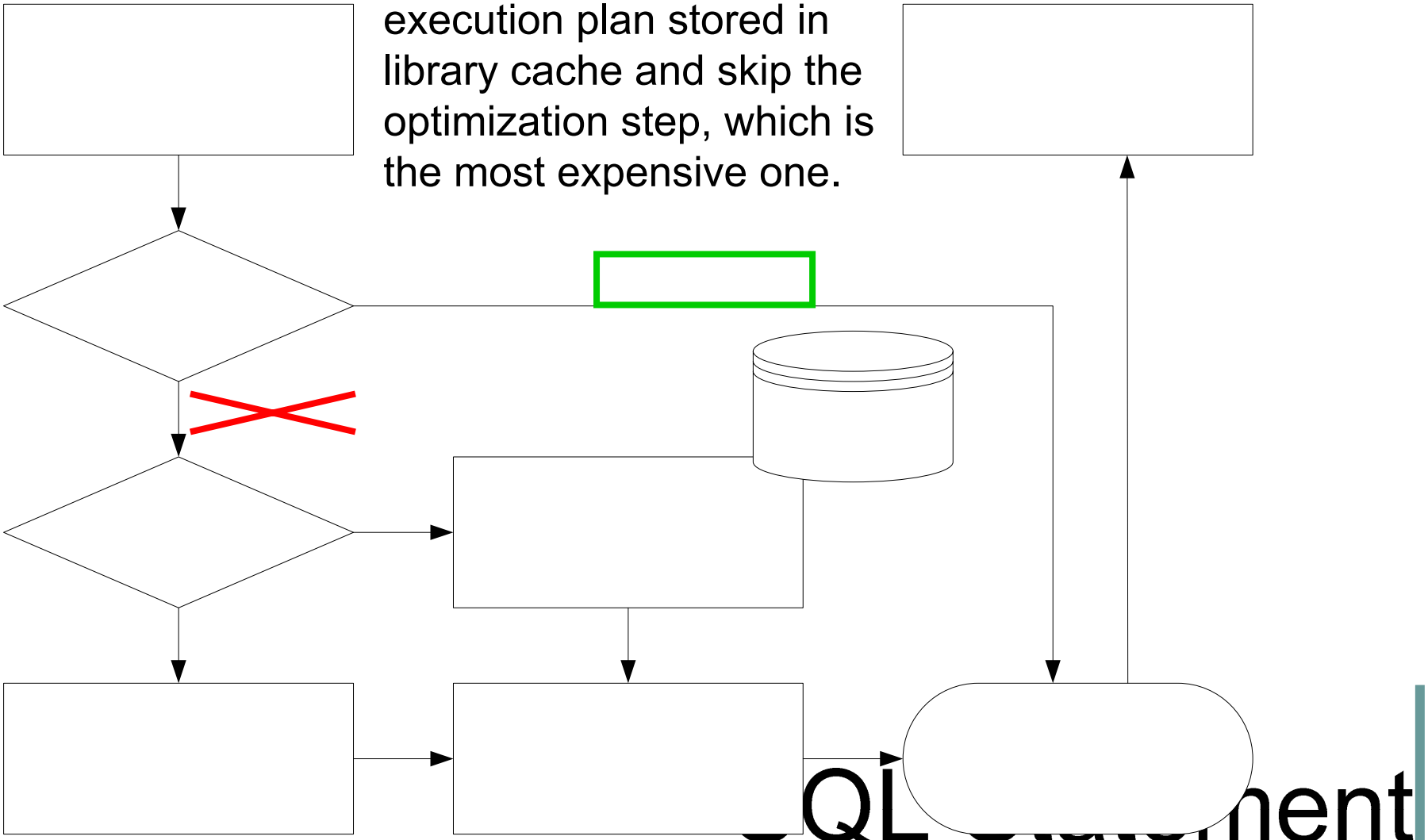
- But why would you try to outsmart the optimizer?
- Consider using: `FIRST_ROWS`, `ALL_ROWS` for setting the optimizer goal, or `APPEND` for direct-load nologging inserts (bulk loading).
- Generally avoid!

Oracle memory structures



Avoid hard parsing...

Soft parse lets you reuse execution plan stored in library cache and skip the optimization step, which is the most expensive one.



CERN School of Computing

...it's easier to...

```
String myName = "O'Really";
String sql =
 "select sal from emp where ename = '"+myName+"'";
Statement stmt = conn.createStatement(sql);
ResultSet rs = stmt.executeQuery(sql);
```

```
String sql =
   "select sal from emp where ename =
   '"+myName.replaceAll("'","''")+"'";
```

```
String myName = "O'Really";
String sql =
  "select sal from emp where ename = ?";
PreparedStatement stmt = conn.prepareStatement(sql);
stmt.setString(1, myName);
ResultSet rs = stmt.executeQuery();
```

CERN School of Computing

...use bind variables!

- Bind variables reduce the number of hard parses and therefore greatly improve scalability of your software.
- It's less secure to code without them (sql injection)!
- It's actually easier to code using bind variables.

There's hardly any rule without exceptions. A literal inside your sql query may provide extra information to the optimizer. If your query takes minutes to execute, then a hard parse does not really make a difference.

Execution plans – how to read them?

- Create `plan_table` first: `$ORACLE_HOME/rdbms/admin/utlxplan.sql`
- Use `explain plan` to store execution plan into `plan_table`
- Use `dbms_xplan` to print execution plan in a readable way (`utlxpls.sql`):

```
SET LINESIZE 130
SET PAGESIZE 0
select * from table(DBMS_XPLAN.DISPLAY);
```

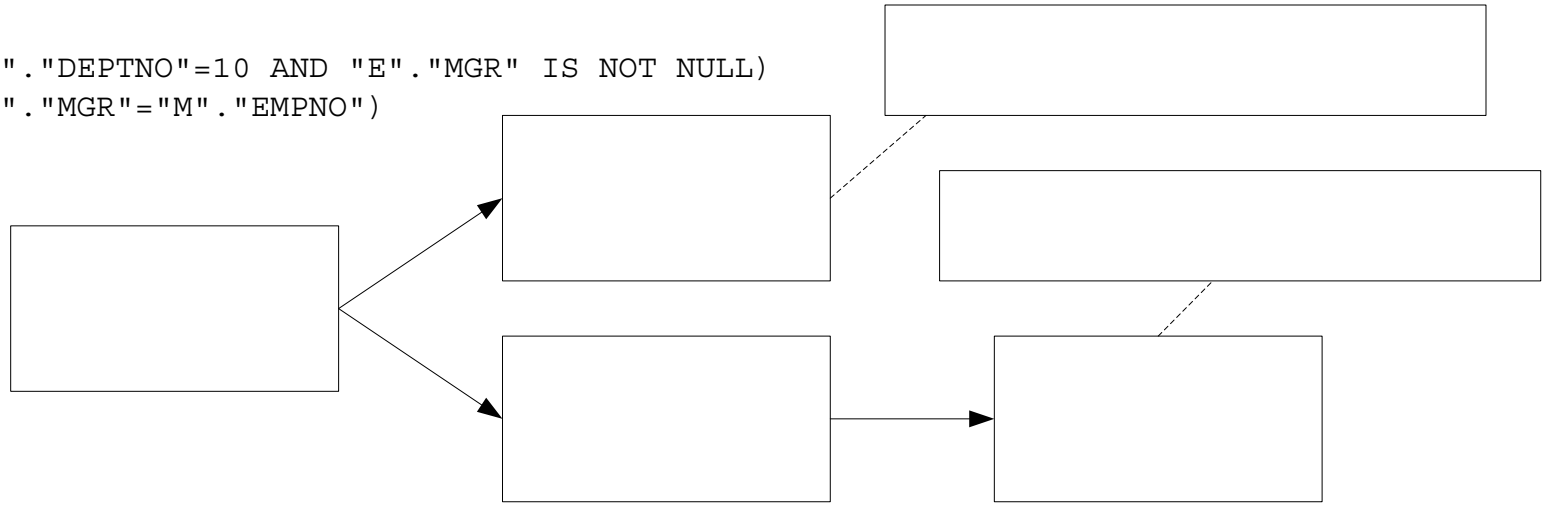
Execution plans – how to read them?

```
Connected to:
Oracle9i Enterprise Edition Release 9.2.0.6.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.6.0 - Production

DEVDB:SQL> explain plan for select e.ename emp, m.ename mgr
  2  from emp e, emp m
  3  where e.mgr = m.empno
  4  and e.deptno = 10;

Explained.

DEVDB:SQL> select * from table(dbms_xplan.display);

------------------------------------------------------------------------------------
| Id  | Operation                     |  Name          | Rows  | Bytes | Cost (%CPU)|
------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                |     3 |    69 |    12   (9)|
|   1 |  NESTED LOOPS                 |                |     3 |    69 |    12   (9)|
|*  2 |   TABLE ACCESS FULL           | EMP            |     3 |    39 |     9  (12)|
|   3 |   TABLE ACCESS BY INDEX ROWID| EMP            |     1 |    10 |     2  (50)|
|*  4 |    INDEX UNIQUE SCAN          | EMP_EMPNO_PK   |     1 |       |            |
------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter("E"."DEPTNO"=10 AND "E"."MGR" IS NOT NULL)
   4 - access("E"."MGR"="M"."EMPNO")
```

Execution plans – how to read them?

```
select e.ename emp, m.ename mgr
from tuneemp e, tuneemp m
where e.mgr = m.empno and e.deptno = 10;
```

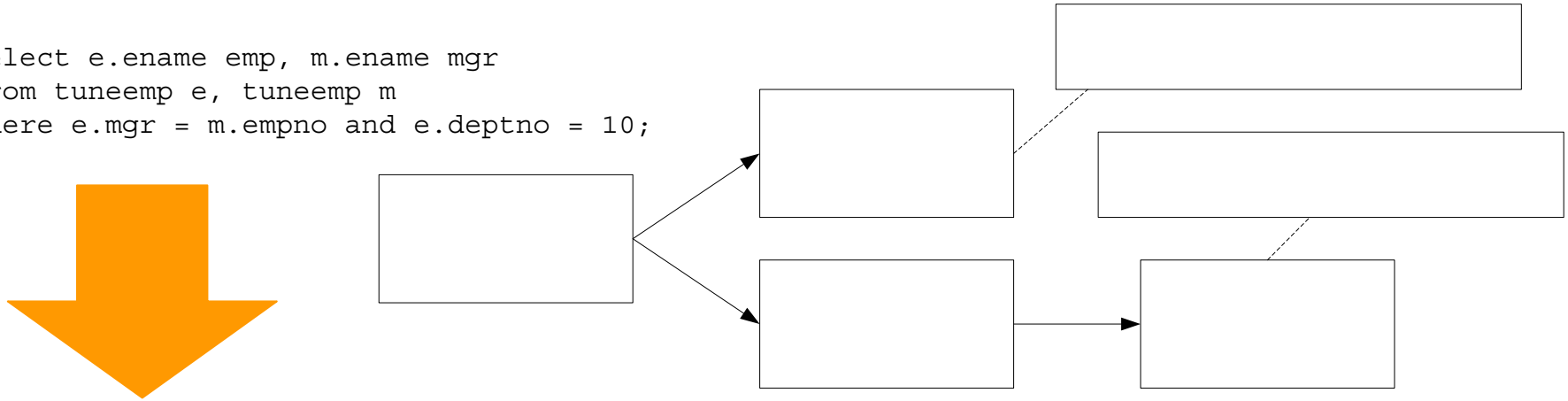
```
-------------------------------------------------------------------------------
| Id  | Operation                     |  Name          | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                |     3 |    69 |    12   (9)|
|   1 |  NESTED LOOPS                 |                |     3 |    69 |    12   (9)|
|*  2 |   TABLE ACCESS FULL           | EMP            |     3 |    39 |     9  (12)|
|   3 |   TABLE ACCESS BY INDEX ROWID| EMP            |     1 |    10 |     2  (50)|
|*  4 |    INDEX UNIQUE SCAN          | EMP_EMPNO_PK   |     1 |       |            |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter("E"."DEPTNO"=10 AND "E"."MGR" IS NOT NULL)
   4 - access("E"."MGR"="M"."EMPNO")
```



CERN School of Computing

Execution plans – how to read them?

```
select e.ename emp, m.ename mgr
from tuneemp e, tuneemp m
where e.mgr = m.empno and e.deptno = 10;
```



For each row r1 in

(select * from emp where deptno=10 and mgr is not null)

Loop

Find rowid of row r2 using index emp_empno_pk;

Get row r2 by rowid;

Output r1.ename, r2.ename;

End loop

Use appropriate tools – autotrace

- Explain plan shows the plan without executing the statement. The statistics are estimates used to prepare the plan, not real values.
- To see real execution statistics and the plan of the statement you have just executed in sql*plus, use autotrace.
- Turn it on using

```
set autotrace on
[explain|statistics|traceonly]
```

- Remember both explain plan and autotrace show you execution plan for the current state of the database. Different plans might have been used in the past!

Use appropriate tools – autotrace

```
Connected to:
Oracle9i Enterprise Edition Release 9.2.0.6.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.6.0 - Production

DEVDB:SQL> set autotrace on
DEVDB:SQL> set timing on
DEVDB:SQL> select e.ename emp, m.ename mgr
  2  from emp e, emp m
  3  where e.mgr = m.empno
  4  and e.deptno = 10;

EMP        MGR
---------- ----------
CLARK      KING
MILLER     CLARK

Elapsed: 00:00:01.16

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=12 Card=3 Bytes=69)
   1    0   NESTED LOOPS (Cost=12 Card=3 Bytes=69)
   2    1     TABLE ACCESS (FULL) OF 'EMP' (Cost=9 Card=3 Bytes=39)
   3    1     TABLE ACCESS (BY INDEX ROWID) OF 'EMP' (Cost=2 Card=1 By
           tes=10)

   4    3       INDEX (UNIQUE SCAN) OF 'EMP_EMPNO_PK' (UNIQUE) (Cost=1
           Card=1)
```

Use appropriate tools – autotrace

Number of SQL statements executed in order to execute your SQL statement

Total number of blocks read from the buffer cache in current mode

```
Statistics
----------------------------------------------------------
        399  recursive calls
          0  db block gets
         95  consistent gets
          5  physical reads
          0  redo size
        478  bytes sent via SQL*Net to client
        500  bytes receiv   via SQL*Net from client
          2  SQL*Net round    to/from client
          8  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

Number of times a consistent read was requested for a block in the buffer cache. Consistent reads may require read asides to the undo (rollback) information and these reads will be also counted here

Number of physical reads from the datafiles into the buffer cache

Use appropriate tools – tkprof

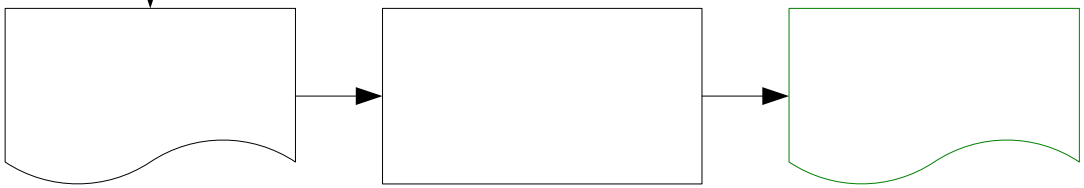
- Use tkprof to analyze trace files
- Enable trace using:

```
alter session set timed_statistics=true;
alter session set sql_trace=true;
```

- Trace files are stored on the database server
- At CERN, you can use:

```
DEVDB:SQL> execute cern_trace.cstart_trace;
... statements ...
DEVDB:SQL> execute
cern_trace.cstop_trace('your.name@cern.ch');
```

Use appropriate tools – tkprof



```
********************************************************************************

select e.ename emp, m.ename mgr
from emp e, emp m
where e.mgr = m.empno
and e.deptno = 10

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.02       0.02          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.01          7         12          0           2
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.02       0.04          7         12          0           2

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 1091

Rows     Row Source Operation
-------  ---------------------------------------------------
      2  NESTED LOOPS
      2   TABLE ACCESS FULL EMP
      2   TABLE ACCESS BY INDEX ROWID EMP
      2    INDEX UNIQUE SCAN EMP_EMPNO_PK (object id 236407)

********************************************************************************
```

alter session set …statement sql_trace=true;

Use appropriate tools – tkprof

You might also consider using:

```
alter session set events
  '10046 trace name context forever, Level N'
```

where *N* can be:

- 1 to enable the standard SQL_TRACE facility,
- 4 to enable SQL_TRACE and also capture bind variable values,
- 8 to enable SQL_TRACE and also capture wait events,
- 12 to enable standard SQL_TRACE and also capture bind variables and wait events.

CERN School of Computing

Use appropriate tools – your own tools inside your code

Get ready for future performance problems.

Consider:

- logging and timing statements that can be turned on/off on demand
- surrounding your code with

```
alter session set sql_trace=true;
alter session set sql_trace=false;
```

that can be turned on/off on demand

Design to perform

- Avoid „let's build it first, we'll tune it later" attitude.
- Optimize to your most frequent type of query.
- There's more than one type of table:
  - Heap (standard) tables
  - B*Tree index clusters
  - Hash clusters
  - Index Organized Tables
- and more than one type of index:
  - B*Tree (standard) indexes
  - Function based indexes
  - Bitmap indexes
  - Domain indexes

CERN School of Computing

Desing to perform – B*Tree index clusters

- B*Tree index cluster physically collocates data by a common key.
- The data is not sorted; it's just physically stored together.
- It uses a B*Tree index to store a key value and block address where the data can be found.
- It allows you to store data from multiple database tables in the same physical database block.

- You cannot do direct-path loading into a cluster.
- You cannot partition clustered tables.
- You need to control the way the data is loaded.

Design to perform – B*Tree index clusters

```
Connected to:
Oracle9i Enterprise Edition Release 9.2.0.6.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.6.0 - Production

DEVDB:SQL> create cluster emp_dept_cluster_btree
  (deptno number(2)) size 50;
Cluster created.

DEVDB:SQL> create index emp_dept_cluster_id on cluster
emp_dept_cluster_btree;
Index created.

DEVDB:SQL> create table dept (
2     deptno number(2) primary key,
3     dname varchar2(14),
4     loc varchar2(13)
5   ) cluster emp_dept_cluster_btree (deptno);
Table created.

DEVDB:SQL> create table emp (
  2    empno number(4) primary key,
  3    ename varchar2(10),
...
  9    deptno number(2) not null,
 10    foreign key (deptno) references dept
 11  ) cluster emp_dept_cluster_btree(deptno);
Table created.
```

Desing to perform – hash clusters

- Hash cluster uses a hashing algorithm to convert the key value into a database block address, thus bypassing all I/O except for the block read itself.
- Optimally, there will be one logical I/O used to perform a lookup.
- Consider using a single-table hash cluster for lookup tables!

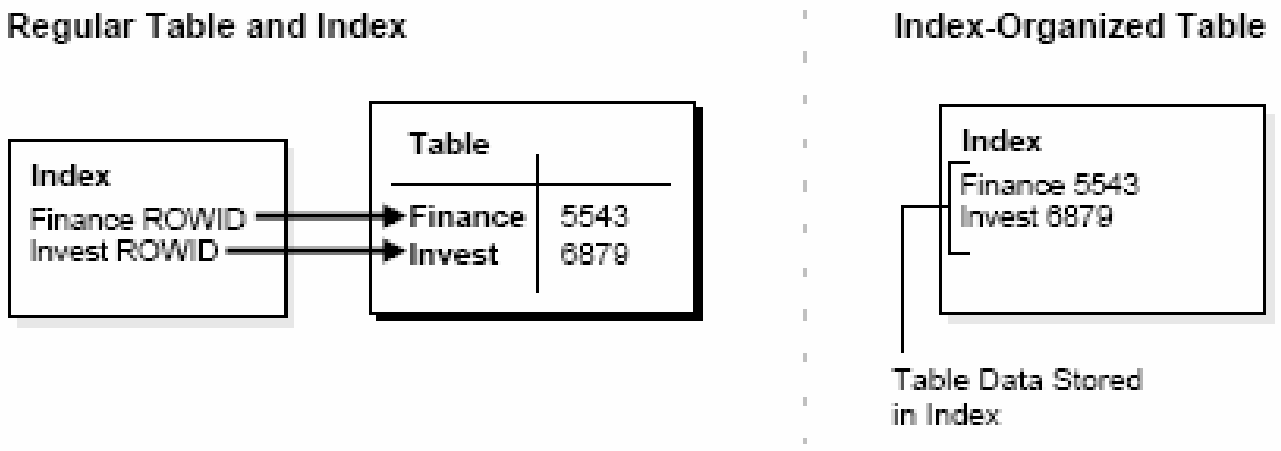
```
create cluster dept_cluster_hash
  (deptno number(2)) hashkeys 100 size 50;
```

- It is an issue to correctly size both types of clusters

CERN School of Computing

Design to perform – Index Organized Tables

- IOT is simply a table stored in an index.
- The data is sorted by key.
- It is very useful for association tables (used in many-to-many relationships).



- Slower to insert into than regular tables

Design to perform – function based indexes

- Perfect for case-insensitive searches or sorts
- Enable searching on complex equations or equations using your own functions
- Let you implement
  - selective indexing
  - selective uniqueness

```
create index emp_lower_ename
on emp (lower(ename));
```

Design to perform – bitmap indexes

- Used for low-cardinality columns
- Good for multiple where conditions (logical bit-wise operations can be used to combine bitmaps)
- Use minimal storage space
- Good for very large tables

```
create bitmap index emp_ix on emp (deptno));
```

- Updates to key columns are very expensive
- Not suitable for OLTP applications with large number of concurrent transactions modifying the data

Design to perform - domain indexes

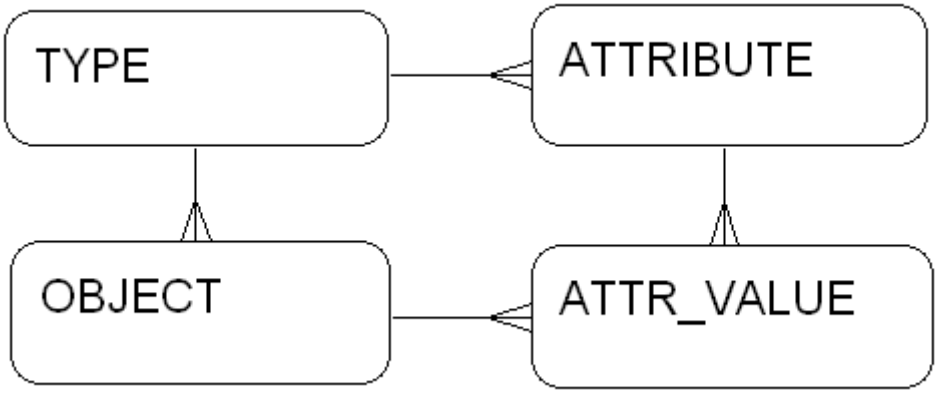
- Extensible indexing
- Allow third-party company to create new index type
- Enable indexing customized complex data types such as documents or spatial data
- Most popular: Oracle Text (Intermedia):

```
create index emp_cv on emp(cv)
indextype is ctxsys.context;

select * from emp where contains
(cv, 'oracle near tuning WITHIN PARAGRAPH')>0;
```

Don’t be too generic

Careful with:

- generic data models



- excessive column sizes „just in case”
- database abstraction layers
- database independency

Test before going into production

- Check how your application performs under stress,
- with 10, 100, 1000 users (concurrency)
- doing real work.
- Be careful about stubbed out API's.
- Keep your tests for the future.

CERN School of Computing

Exercises

Ex. 1. Checking execution plans

Ex. 2. Managing statistics

Ex. 3. Using indexes

Ex. 4. Bind variables

Ex. 5. Autotrace and tuning problems

Look for tuning_exercises.zip on CD.

References

- http://oradoc/
  - Concepts
  - Performance Tuning Guide and Reference
  - ...
- Tom Kyte's
  - „Effective Oracle by Design"
  - http://asktom.oracle.com
  - http://computing-colloquia.web.cern.ch/computing-colloquia/past.htm#2005
- CERN Database Tutorials & workshop materials