🚀 *Lecture 4 - BDA 05-10-2018*
Database Administration 🚀

1. **More advanced SQL**
   Windows function - Rank() Over() - With
2. **Overview Admin DBA**
   Goal - Skills - Difficulties - Tools
3. **Performance Optimization**
   SQL optimizer - EXPLAIN - **Indexes** - Cluster
   Caching - Vocabulary - Partitions - Pooling
4. **Logical / Physical Structure Organization**
   Data files - Tablespaces - Segments - Extents
   - Data Blocks - Storage Clause

ORACLE

Bibliography:
https://docs.oracle.com

More Advanced SQL

When I discover SQL is turing compliant

http://lesjoiesducode.fr

RANK() OVER() + ROW_NUMBER() + DENSE_RANK()

```
SELECT
  empno,
  ROW_NUMBER()  OVER (ORDER BY empno) as row_number,
  RANK()        OVER (ORDER BY empno) as rank,
  DENSE_RANK()  OVER (ORDER BY empno) as dense_rank
FROM emp
ORDER BY 1, 2
```

	EMPNO	ROW_NUMBER	RANK	DENSE_RANK
1	7369	1	1	1
2	7499	2	2	2
3	7521	3	3	3
4	7566	4	4	4
5	7654	5	5	5
6	7698	6	6	6
7	7782	7	7	7
8	7788	8	8	8
9	7844	9	9	9
10	7876	10	10	10
11	7900	11	11	11
12	7902	12	12	12
13	7934	13	13	13

RANK() OVER() + ROW_NUMBER() + DENSE_RANK()

```
SELECT
  v,
  ROW_NUMBER()  OVER (ORDER BY v) as row_number,
  RANK()        OVER (ORDER BY v) as rank,
  DENSE_RANK()  OVER (ORDER BY v) as dense_rank
FROM my_table
ORDER BY 1, 2
```

my_table

V	ROW_NUMBER
a	1
a	2
a	3
b	4
c	5
c	6
d	7
e	8

Result of query:

V	ROW_NUMBER	RANK	DENSE_RANK
a	1	1	1
a	2	1	1
a	3	1	1
b	4	4	2
c	5	5	3
c	6	5	3
d	7	7	4
e	8	8	5

Filter On RANK

- It's not possible to use having:

```
SELECT content,
      ROW_NUMBER() OVER(ORDER BY content),
      RANK() OVER(ORDER BY content) AS rank,
      DENSE_RANK() OVER(ORDER BY content)
      FROM emp
      HAVING rank > 1
```

- You have to create an inner query and filter in the upper query.

```
SELECT * FROM  (
      SELECT content,
            ROW_NUMBER() OVER(ORDER BY content),
            RANK() OVER(ORDER BY content) AS rank,
            DENSE_RANK() OVER(ORDER BY content)
            FROM emp
      )
WHERE rank > 1
```

Partition inside RANK

- You have the rank inside a group with PARTITION:

```
SELECT EFIRST, ENAME, DEPTNO,
    RANK() over (partition by DEPTNO order by EMPNO desc)
    FROM EMP
```

	EFIRST	ENAME	DEPTNO	RANK()OVER(PARTITIONBYDEPTNOORDERBYEMPNODESC)
1	ALICE	MILLER	10	1
2	JOHN	CLARK	10	2
3	MARIA	FORD	20	1
4	JOSEPH	ADAMS	20	2
5	GUY	SCOTT	20	3
6	JOHN	JONES	20	4
7	JOHN	SMITH	20	5
8	ALAN	JAMES	30	1
9	PETER	TURNER	30	2
10	BOB	BLAKE	30	3
11	JOE	MARTIN	30	4
12	PETER	WARD	30	5
13	BOB	ALLEN	30	6

3 best salaries in each dept

```
SELECT * FROM (
    SELECT EFIRST, ENAME, SAL, DEPTNO,
        RANK() over (partition by DEPTNO order by SAL desc) as rank
        FROM EMP
)
WHERE rank <= 3
```

	EFIRST	ENAME	SAL	DEPTNO	RANK
1	JOHN	CLARK	2450	10	1
2	ALICE	MILLER	1300	10	2
3	GUY	SCOTT	3000	20	1
4	MARIA	FORD	3000	20	1
5	JOHN	JONES	2975	20	3
6	BOB	BLAKE	2850	30	1
7	BOB	ALLEN	1600	30	2
8	PETER	TURNER	1500	30	3

More Advanced SQL :

RANK, RANK_DENSE, ...
Are
Analytical Queries

Analytic Function

- Syntax:

FUNCTION_NAME(**column** | expression, **column** | expression,...) OVER (**Order-by**-Clause)

- Function name can be:

Sum, avg, rank,....

- Use as an attribute:

SELECT
FUNCTION_NAME (**column** | expression...)
OVER (**Order-by**-Clause)
FROM table

- Example:

SELECT empno, deptno, sal,
SUM(sal) OVER
(PARTITION **BY** deptno **ORDER BY** empno)
AS sum
FROM emp

	EMPNO	DEPTNO	SAL	SUM
1	7782	10	2450	2450
2	7934	10	1300	3750
3	7369	20	800	800
4	7566	20	2975	3775
5	7788	20	3000	6775
6	7876	20	1100	7875
7	7902	20	3000	10875
8	7499	30	1600	1600
9	7521	30	1250	2850
10	7654	30	1250	4100
11	7698	30	2850	6950
12	7844	30	1500	8450
13	7900	30	950	9400

SUM
SUM
SUM

Choosing window

- Inside over:
  - **ROWS N** FOLLOWING
  - **ROWS N** PRECEDING
  - **BETWEEN N** PRECEDING AND **N** FOLLOWING

	EMPNO	HIREDATE	SAL	MIN
1	7369	17/12/80	800	800
2	7499	20/02/81	1600	800
3	7521	22/02/81	1250	1250
4	7566	02/04/81	2975	1250
5	7698	01/05/81	2850	2450
6	7782	09/06/81	2450	1500
7	7844	08/09/81	1500	1250
8	7654	28/09/81	1250	950
9	7900	03/12/81	950	950
10	7902	03/12/81	3000	950
11	7934	23/01/82	1300	1300
12	7788	09/12/82	3000	1100
13	7876	12/01/83	1100	1100

- Example:

```
SELECT empno, deptno, sal, hiredate,
    MIN (sal) OVER (ORDER BY HIREDATE
        ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS min
FROM emp
```

CASE WHEN

- Syntax

```
SELECT
    CASE
        WHEN boolean1 THEN Result1
        WHEN boolean2 THEN Result2
        ELSE ResultN
    END
FROM table
```

- Example:

```
SELECT efirst, ename, sal,
  CASE    WHEN sal < 1000 THEN 'Low'
          WHEN sal > 2000 THEN 'High'
          ELSE 'Medium' END
  AS salaryLevel
FROM emp;
```

	EFIRST	ENAME	SAL	SALARYLEVEL
1	JOHN	SMITH	800	Low
2	BOB	ALLEN	1600	Medium
3	PETER	WARD	1250	Medium
4	JOHN	JONES	2975	High
5	JOE	MARTIN	1250	Medium
6	BOB	BLAKE	2850	High
7	JOHN	CLARK	2450	High
8	GUY	SCOTT	3000	High
9	PETER	TURNER	1500	Medium
10	JOSEPH	ADAMS	1100	Medium
11	ALAN	JAMES	950	Low
12	MARIA	FORD	3000	High
13	ALICE	MILLER	1300	Medium

Window With CASE WHEN

- Inside the over you can use:

```sql
SELECT CASE WHEN sal = min then 'Minimum' else 'Not a min' END as diff,
    empno, deptno, hiredate,sal, min from (
        SELECT empno, deptno, sal, hiredate,
            MIN(sal) OVER (ORDER BY HIREDATE ROWS
                BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS min
            FROM emp
    )
```

	DIFF	EMPNO	DEPTNO	HIREDATE	SAL	MIN
1	Minimum	7369	20	17/12/80	800	800
2	Not a min	7499	30	20/02/81	1600	800
3	Minimum	7521	30	22/02/81	1250	1250
4	Not a min	7566	20	02/04/81	2975	1250
5	Not a min	7698	30	01/05/81	2850	2450
6	Not a min	7782	10	09/06/81	2450	1500
7	Not a min	7844	30	08/09/81	1500	1250
8	Not a min	7654	30	28/09/81	1250	950
9	Minimum	7900	30	03/12/81	950	950
10	Not a min	7902	20	03/12/81	3000	950
11	Minimum	7934	10	23/01/82	1300	1300
12	Not a min	7788	20	09/12/82	3000	1100
13	Minimum	7876	20	12/01/83	1100	1100

Too much inner functions

What if we want to select only persons with diff equal to “Minimum”. It would give:

```
SELECT * FROM (
      SELECT CASE WHEN sal = min then 'Minimum' else 'Not a min' END as diff,
            empno, deptno, hiredate,sal, min from (
                  SELECT empno, deptno, sal, hiredate,
                        MIN(sal) OVER (ORDER BY HIREDATE ROWS
                              BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS min
                        FROM emp
            )

) WHERE diff = ‘Minimum’
```

⚠️ This gives a lot of INNER queries (3 nested SELECT).
In real life, It’s very easy to have queries with much more nested queries.

WITH clause to the rescue 🦸 !

❖ The WITH clause in SQL was introduced in standard SQL to simplify complex long queries, especially those with JOINs and subqueries. Often interchangeably called CTE or subquery refactoring, a WITH clause defines a temporary data set whose output is available to be referenced in subsequent queries.

SYNTAX:

```
WITH
    table1 AS (SELECT …….. ),
    table2 AS (SELECT …….. ),
    table3 AS (SELECT …….. )

SELECT * FROM
    table2, table3
    …
```

In the query, you can re-use previous table1.

WITH clause to the rescue 🦸 !

- We can simplify readability of previous function like this:

```
WITH  emp_with_min_sal_over_window AS
      (SELECT empno, deptno, sal, hiredate,
            MIN(sal) OVER (ORDER BY HIREDATE ROWS
                              BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS min
                              FROM emp
      ),
      emp_with_is_minimum_information AS
      (SELECT CASE WHEN sal = min then 'Minimum' else 'Not a min' END as diff,
             empno, deptno, hiredate,sal, min FROM emp_with_min_sal_over_window
      )
SELECT * FROM emp_with_is_minimum_information
      WHERE diff = ‘Minimum’
```

Preparation / staging / temporary

Final result

More Advanced SQL

SELECTIVE AGGREGATE

SELECTIVE AGGREGATE

Imagine you have aggregate on different condition:

- get sum of salary all employee
- get sum of salary all employee whose hiredate > 1982
- get average of salary all employee
- get number of employee in dept 20 and 30

Any body who don't know aggregate / filtering will write 4 queries:

- SELECT SUM(sal) FROM EMP
- SELECT SUM(sal) FROM EMP WHERE hiredate > 1982
- SELECT AVG(sal) FROM EMP
- SELECT AVG(sal) FROM EMP WHERE deptno = 20 or deptno = 30

But this means 4 Full Scan ! Not performant and high cost.

On some system, you per volumetry read. (ie BigQuery is 5eur/To)

SELECTIVE AGGREGATE

You can use FILTERING on AGGREGATE which would do all in query:

```
SELECT
        SUM(sal),
        SUM(sal) FILTER (WHERE hiredate > 1982),
        AVG(sal),
        COUNT(*) FILTER (WHERE deptno = 20 or deptno = 30)
    FROM EMP
```

You divide cost by the number of queries ! (here cost divided by 4)

More and more

Why ?

http://lesjoiesducode.fr

Queries everywhere

```
SELECT
        user_id,
        COUNT(DISTINCT(boat_id)) * 100
        /
        (SELECT COUNT(*) FROM Boats)
    FROM Reserve
    GROUP BY user_id
```

What does the query is doing ?

As you see you can do queries almost everywhere.

Built-in Functions

Use built-in Functions

In SQL, every database offers a set of built-in functions:

- Utils for Date (EXTRACT, date_part ...)

SELECT EXTRACT(YEAR FROM order_date) AS year FROM orders -- If order_date is 2024-01-01, results is 2024

- JSON parsing (jsonb_path_query,)

Example here: https://www.postgresql.org/docs/current/functions-json.html

- String functions (Substring, rtrim, concat, upper ...)

Example here: https://www.postgresql.org/docs/current/functions-string.html

- Geometric Functions (add/subtract coordinates ...)

Example here: https://www.postgresql.org/docs/current/functions-geometry.html

More:

https://www.postgresql.org/docs/current/functions.html

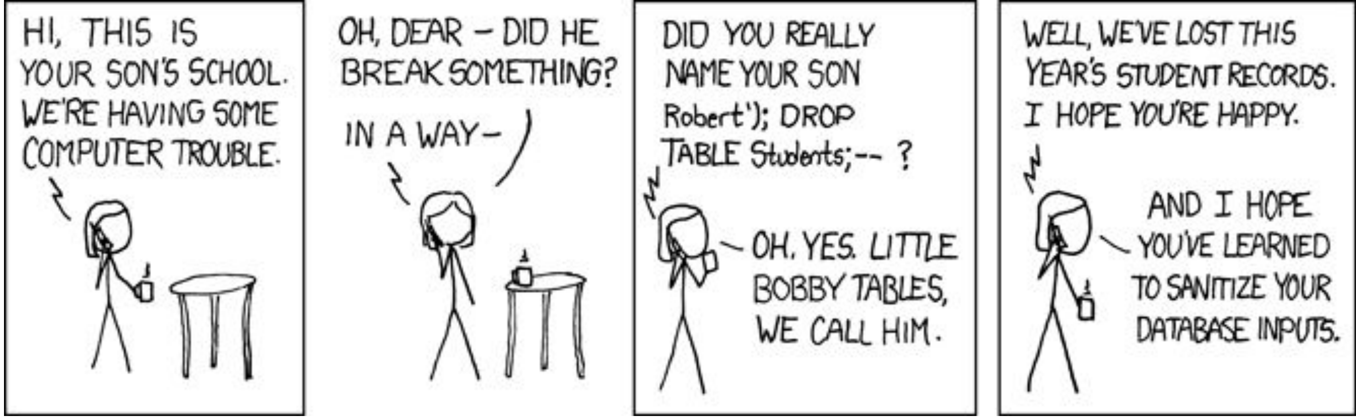
Extremely advanced SQL

- This blog post Online gradient descent written in SQL shows how to do machine learning in SQL: https://maxhalford.github.io/blog/ogd-in-sql/?ref=limit-5

- Note the use of "WITH RECURSIVE" : https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/with.html

Overview Admin DBA

Goal



- Installing and upgrading the Oracle Database server and application tools
- Allocating system storage and planning future storage requirements for the database system
- Creating primary database storage structures (tablespaces) after application developers have designed an application
- Creating primary objects (tables, views, indexes) once application developers have designed an application
- Modifying the database structure, as necessary, from information given by application developers
- Enrolling users and maintaining system security
- Ensuring compliance with Oracle license agreements
- Controlling and monitoring user access to the database
- Monitoring and optimizing the performance of the database
- Planning for backup and recovery of database information
- Maintaining archived data on tape
- Backing up and restoring the database
- Contacting Oracle for technical support

Skills

- Communication skills
- Knowledge of database theory / design
- Knowledge about the RDBMS itself, e.g. Microsoft SQL Server or MySQL
- Knowledge of structured query language (SQL), e.g. SQL/PSM or Transact-SQL
- General understanding of distributed computing architectures, e.g. Client–server model
- General understanding of operating system, e.g. Windows or Linux
- General understanding of storage technologies and networking
- General understanding of routine maintenance, recovery of a database

Difficulties

- **The position requires a broad spectrum of knowledge** (development, system administration and even management).
- **The consequences of failure are usually greater for a DBA** than a developer.
- **The better a DBA does their job the less visibility they have.** A DBA with a database that is secure, recoverable, available, and performing well will lack recognition. DBAs get noticed when there are problems.

When I Have to make a delete in Prod

http://lesjoiesducode.fr

When I realize that it's not in base "test" that i made the delete but in Prod

http://lesjoiesducode.fr

Tools

Oracle Tools:

- SQL Developer
- SQL*Plus *(command line: sqlplus /nolog)*
- Oracle Enterprise Manager Database Control

Others:

- Data modeling tools *(Database Workbench)*
- Sophisticated performance monitoring and alerting tools
- Benchmarking tools
- Advanced query tuning tools
- Backup compression and highly granular recovery tools
- Source code and change management tools *(Git, svn…)*

Performance Optimization

Performances Optimization
Indexes

Indexes (Definition)

- An index is a schema object that contains an entry for each value that appears in the indexed column(s) of the table or cluster and provides direct, fast access to rows. Oracle Database supports several types of index:
  - Normal indexes. (By default, Oracle Database creates **B-tree** indexes.)
  - **Bitmap** indexes, which store rowids associated with a key value as a bitmap
  - **Partitioned** indexes
  - **Function-based** indexes
  - **Domain** indexes

Indexes (Syntax)

- CREATE [ONLINE|OFFLINE]
  [UNIQUE|FULLTEXT|SPATIAL|BITMAP] INDEX
  index_name
  [index_type]
  ON tbl_name (index_col_name,...)
  [index_option] ...

Indexes (B-Tree)

A B-tree is a self-balancing tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in **logarithmic** time. log(n).

=> Use it when you need to find few rows in large quantity of data.

- Example with phone book with 10ˆ6 entries:
  - Stored in normal table:

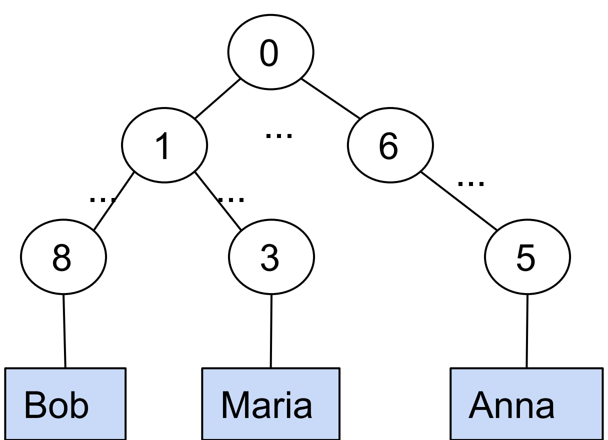
    ["0145013298:Bob", "0145346523:Maria", ..., "064532365:Anna"]

    If you want to find the owner of number 0143543454, you have to go trough all the table, 10ˆ6 operations. Lookup is in O(n).
  - Stored in a B Tree;

    To find a number you need to follow 10 branches.

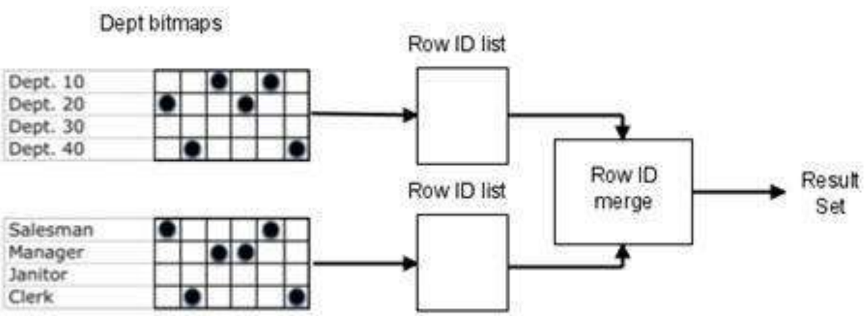
    It's only 10 operations max.

    log(n) operations



Indexes (BitMaps)

- In bitmap structures, a two-dimensional array is created with one column for every row in the table being indexed. Each column represents a distinct value within the bitmapped index.
- Use with high cardinality: When there are 100+ rows for each distinct value.
- Use with Low DML (update/delete/insert).



		Bitmaps			
ROW ID	Hair Colour	Brown	Black	Blonde	Red
1	Brown	1	0	0	0
2	Red	0	0	0	1
3	Brown	1	0	0	0
4	Black	0	1	0	0
5	Blonde	0	0	1	0
6	Brown	1	0	0	0
7	Blonde	0	0	1	0

Indexes (Guideline)

- Consider indexing keys that appear frequently in WHERE clauses.
- Consider indexing keys that frequently join tables in SQL statements. For more information on optimizing joins, see the "Using Hash Clusters for Performance".
- Choose index keys that have high selectivity. The selectivity of an index is the percentage of rows in a table having the same value for the indexed key. An index's selectivity is optimal if few rows have the same value.

Indexes (Guideline)

- Do not use standard B-tree indexes on keys or expressions with few distinct values. (Use Bitmap unless index is modified frequently).
- Do not index frequently modified columns. UPDATE statements that modify indexed columns and INSERT and DELETE statements that modify indexed tables take longer than if there were no index.
- Do not index keys that appear only in WHERE clauses with functions or operators.
- Consider indexing foreign keys of referential integrity constraints in cases in which a large number of concurrent INSERT, UPDATE, and DELETE statements access the parent and child tables. Such an index allows UPDATEs and DELETEs on the parent table without share locking the child table.
- When choosing to index a key, consider whether the performance gain for queries is worth the performance loss for INSERTs, UPDATEs, and DELETEs and the use of the space required to store the index. (Measure)

Indexes (Exercise)

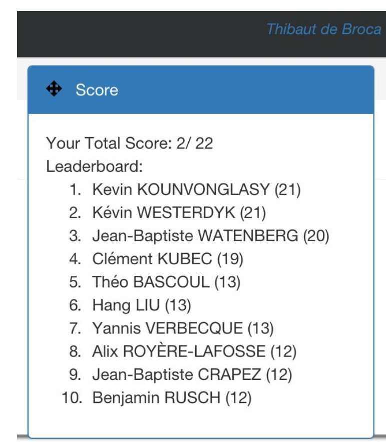
❖ Exercise: How to display the leaderboard of a specific lesson with the following model of tables:

UserApp
idUser (int)
firstName (varchar2)
lastName (varchar2)
mail (varchar2)
password (varchar2)
isTeacher (boolean)

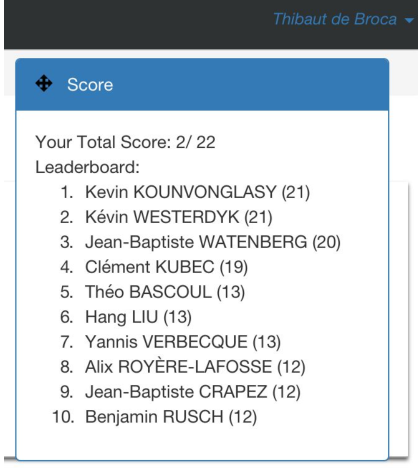
LessonItemAnswer
idLessonItemAnswer (int)
idLessonItem (int)
idUser (int)
answer (text)
answerDate (datetime)
correct (boolean) (0 or 1)

LessonItem
idLessonItem (int)
idLessonItemType (int)
idLesson (varchar2)
title (varchar2)
description (text)
answer (text)
maxPoints (int)
orderItem (int)

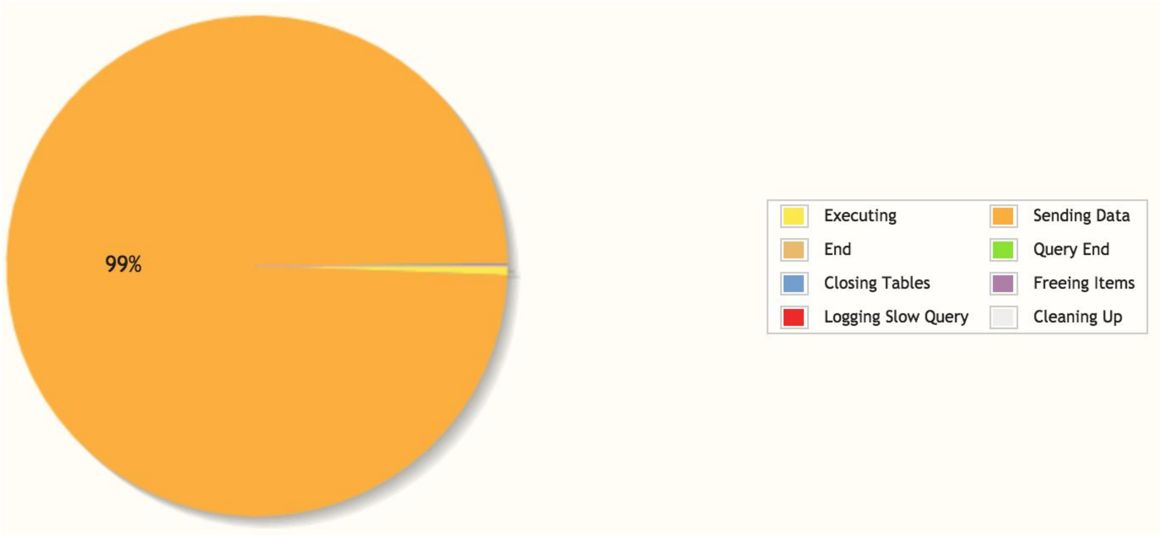
```
SELECT firstName, lastName,
(SELECT SUM(maxPoints) AS score FROM LessonITEM WHERE
      LessonItem.idLessonItem IN
            (SELECT idLessonItem FROM LessonItemAnswer
            WHERE correct AND LessonItemAnswer.idUser = U.idUser
            AND idLesson = {{Input_user}}
            )
) AS score
FROM User U
WHERE !isTeacher
ORDER BY score DESC LIMIT 10
```

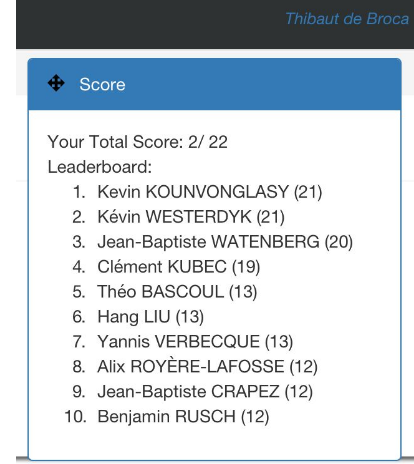


Indexes (Analyze)



- Select COUNT(*) from LessonItem: 60
- Select COUNT(*) from LessonItemAnswer: 5200
- Select COUNT(*) from User: 80
- LeaderboardQuery : 2.4 sec TOO MUCH with a very little number rows ! Plus query time will grow exponentially !
- Explain SELECT ….



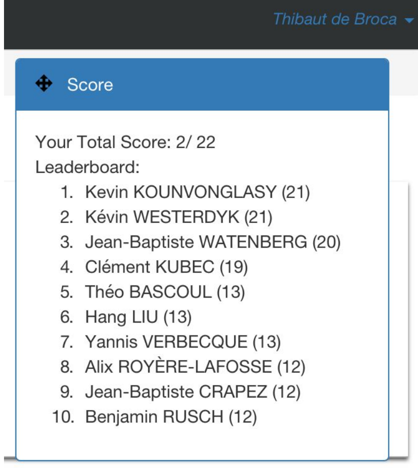


Indexes (Find good Indexes)

- Where can we place some good Indexes ?

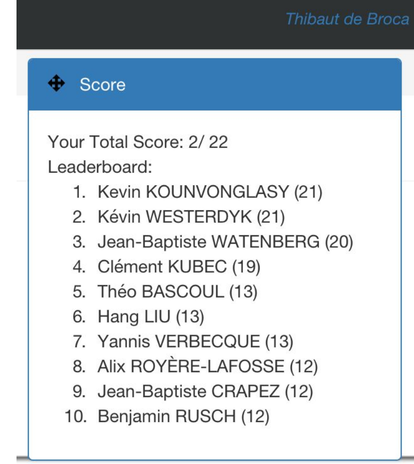
```
SELECT firstName, lastName,
    (SELECT SUM(maxPoints) as score
    FROM LessonItem
    WHERE LessonItem.idLessonItem IN
            (SELECT LessonItem.idLessonItem FROM LessonItem ,
        LessonItemAnswer
            WHERE idLesson = ?
                AND LessonItemAnswer.idLessonItem =
            LessonItem.idLessonItem
            AND idUser = u1.idUser AND correct))
            AS score
    From User u1
    WHERE isTeacher = 0
    ORDER BY score DESC LIMIT 10
```

Indexes (Solution)



- In Blue are the good indexes to place.
  - blue: B-Tree ~~red: BitMap~~
- In green bold are the primary key (implicit indexes).

SELECT firstName, lastName,
(SELECT SUM(maxPoints) as score
FROM LessonItem
WHERE LessonItem.idLessonItem IN
(SELECT LessonItem.idLessonItem FROM LessonItem ,
LessonItemAnswer
WHERE **idLesson** = ?
AND LessonItemAnswer.**idLessonItem** =
LessonItem.**idLessonItem**
AND **idUser** = u1.**idUser** AND **correct**))
AS score
From User u1
WHERE **isTeacher** = 0
ORDER BY score DESC LIMIT 10

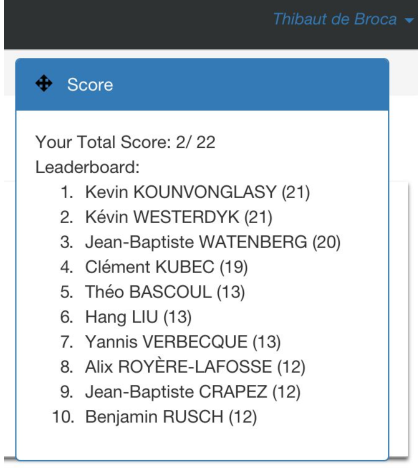


Indexes (add them)

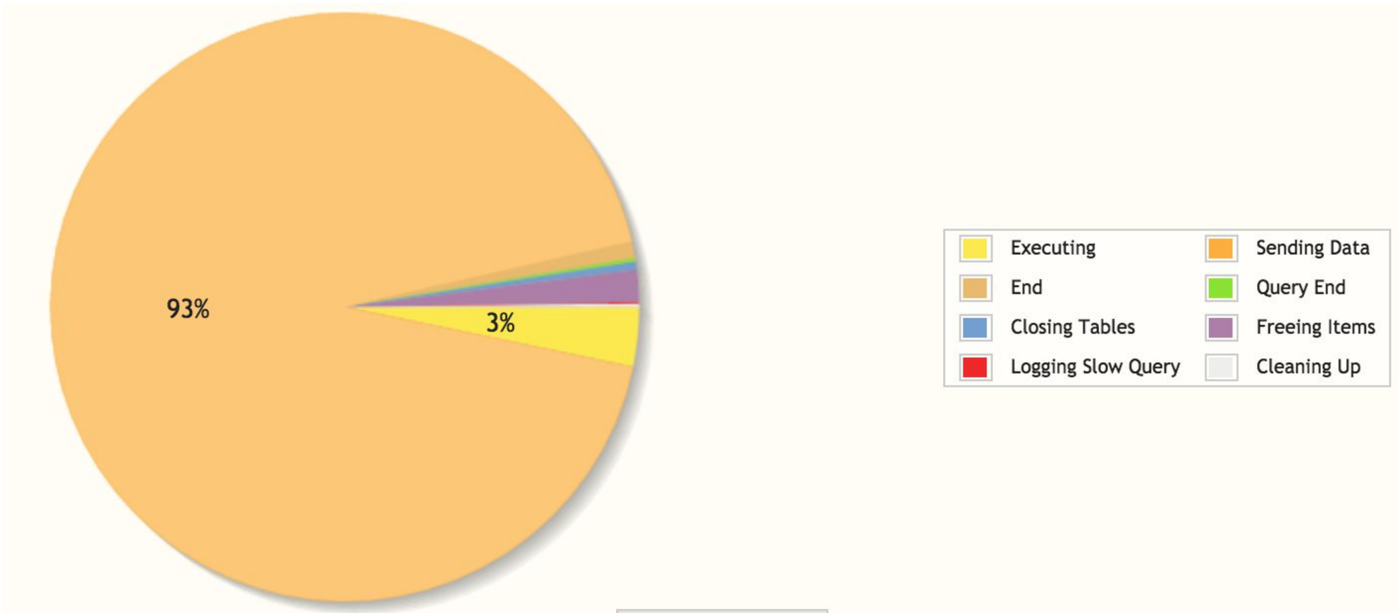
- CREATE INDEX idLesson_index ON LessonItem(idLesson)
- CREATE INDEX idLessonItem_Index ON LessonItemAnswer(idLessonItem)
- CREATE INDEX idUser_index ON LessonItemAnswer(idUser)
- ~~CREATE BITMAP INDEX correct_index ON LessonItemAnswer(correct)*~~
- ~~CREATE BITMAP INDEX isTeacher_index ON User(isTeacher)*~~

*Indexes on Boolean fields are uselss

Indexes (Example)



- We run again the query
- LeaderboardQuery : 3 ms :-), 0.003 sec, 1000 times less than before. Plus time won't grow exponentially. :-D
- Explain SELECT ….



Sargable queries (Search ARGument ABLE)

❖ A query is said to be **sargable** if the DBMS engine can take advantage of an index to speed up the execution of the query.

NON- SARGABLE	SARGABLE
`SELECT * FROM myTable WHERE SQRT(myIntField) > 11.7`	`SELECT * FROM myTable WHERE myIntField > 11.7 * 11.7`
`SELECT * FROM myTable WHERE myNameField LIKE '%Wales%' -- Begins with %, not sargable`	`SELECT * FROM myTable WHERE myNameField LIKE 'Jimmy%' -- Does not begin with %, sargable`
`SELECT * FROM myTable WHERE YEAR(myDate) >= 2023`	`SELECT * FROM myTable WHERE myDate >= 2023-01-01 AND myDate <= 2023-31-12`

How to write Sargable queries

- Avoid functions or calculation on indexed columns on WHERE clause
- Use direct comparison
- If you need to apply a function on a column, create a function-based index

SQL Optimizer

When Someone ask me to run the new database on the previous server

http://lesjoiesducode.fr

SQL Optimizer (Overview)

- The database can execute a SQL statement in multiple ways, such as full table scans, index scans, nested loops, and hash joins.
- The optimizer considers many factors related to the objects and the conditions in the query when determining an execution plan.
- This determination is an important step in SQL processing and can greatly affect execution time.

SQL Statistics (Overview)

- Optimizer statistics are a collection of data that describe more details about the database and the objects in the database. These statistics are used by the query optimizer to choose the best execution plan for each SQL statement. Optimizer statistics include the following:
  - Table statistics, Column statistics, Index statistics, System statistics (I/O, CPU...)
- Automatic statistics gathering is enabled by default (done every night by nightly job).
- For Manual Stats gathering, use DBMS_STATS package.

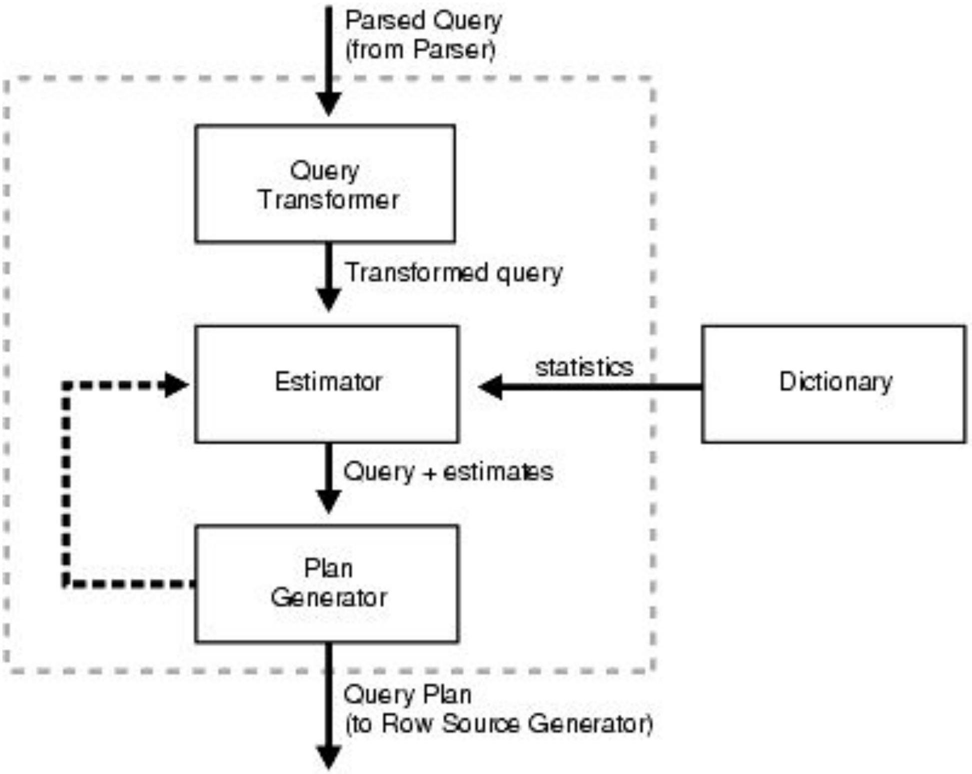
SQL Statistics (Postgre)

- VACUUM [FULL] [FREEZE] [VERBOSE] [table] reclaims storage occupied by dead tuples.
- REINDEX INDEX myindex
  REINDEX The REINDEX command rebuilds one or more indices, replacing the previous version of the index.

SQL Optimizer (Steps)

- <u>Step 1</u>: The optimizer generates a set of potential plans for the SQL statement based on available access paths and hints.
- <u>Step 2</u>: The optimizer estimates the cost of each plan based on statistics in the data dictionary. Statistics include information on the data distribution and storage characteristics of the tables, indexes, and partitions accessed by the statement.
- <u>Step 3</u>: The optimizer compares the plans and chooses the plan with the lowest cost.
- The output from the optimizer is an **execution plan.**

SQL Optimizer (Process)



Performances Optimization
EXPLAIN

EXPLAIN (Definition)

- The EXPLAIN PLAN statement displays execution plans chosen by the Oracle optimizer for SELECT, UPDATE, INSERT, and DELETE statements. A statement's execution plan is the sequence of operations Oracle performs to run the statement.

EXPLAIN (Content)

- The row source tree is the core of the execution plan. It shows the following information:
  - An ordering of the tables referenced by the statement
  - An access method for each table mentioned in the statement
  - A join method for tables affected by join operations in the statement
  - Data operations like filter, sort, or aggregation
- In addition to the row source tree, the plan table contains information about the following:
  - Optimization, such as the cost and cardinality of each operation
  - Partitioning, such as the set of accessed partitions
  - Parallel execution, such as the distribution method of join

EXPLAIN (Example Oracle)

- Oracle:
  - EXPLAIN PLAN FOR SELECT * FROM emp NATURAL JOIN dept NATURAL JOIN salgrade GROUP BY deptno SORT BY empno;
  - SELECT * FROM PLAN_TABLE;

```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------------------------------------------
Plan hash value: 3841029040

-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name      | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop  | IN-OUT | PQ Distrib
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |           |    24 |  3014 |    35   (0)| 00:00:43 |       |        |        |
|   1 |  PX COORDINATOR                    |           |       |       |            |          |       |        |        |
|   2 |   PX SEND QC (ORDER)               | :TQ10003  |    24 |  3014 |    35   (0)| 00:00:31 |       |        | P->S   | QC (ORDER)
|   3 |    VIEW                            |           |       |       |            |          |       |        | PCWP   |
|   4 |     SORT GROUP BY                  |           |       |       |            |          |       |        | PCWP   |
|   5 |      PX RECEIVE                    |           |    24 |  3014 |    35   (0)| 00:00:01 |       |        | PCWP   |
|   6 |       PX SEND RANGE                | :TQ10002  |    24 |  3014 |    35   (0)| 00:00:01 |       |        | PCWP   | RANGE
|*  7 |        HASH JOIN                   |           |    24 |  3014 |    11   (0)| 00:00:01 |       |        | PCWP   |
|   9 |          PX RECEIVE                |           |    24 |  3014 |     9   (0)| 00:00:01 |       |        | PCWP   |
|  10 |           PX SEND BROADCAST        | :TQ10001  |   480 |   64K|     8   (0)| 00:00:01 |       |        | P->P   | BROADCAST
|  11 |            PX BLOCK ITERATOR       |           |   480 |   64K|     8   (0)| 00:00:01 |       |        | PCWP   |
|  12 |             PX BLOCK ITERATOR      |           |   480 |   64K|     8   (0)| 00:00:01 |       |        | PCWP   |
|* 13 |              TABLE ACCESS FULL     | EMP       |   480 |   64K|     8   (0)| 00:00:21 |       |        | PCWP   |
|  14 |           PX BLOCK ITERATOR        |           |     1 |   248 |     4   (0)| 00:00:01 |     1 |      8 | PCWP   |
|* 15 |            TABLE ACCESS FULL       | DEPT      |     8 |   248 |     4   (0)| 00:00:11 |     8 |     14 | PCWP   |
|  16 |       PX PARTITION RANGE SINGLE    |           |    18 |   960 |    18   (0)| 00:00:01 |     4 |      4 | PCWP   |
|  17 |        PX PARTITION HASH JOIN-FILTER|          |    18 |   960 |    18   (0)| 00:00:01 |:BF0000|:BF0000 | PCWP   |
|* 18 |         TABLE ACCESS FULL          | SALGRADE  |    18 |   960 |    18   (0)| 00:00:18 |   KEY |    KEY | PCWP   |
-------------------------------------------------------------------------------------------------------------------------------------

*Note: Query/Explain Plan adjusted per NDA
```

EXPLAIN (Example - Mysql)

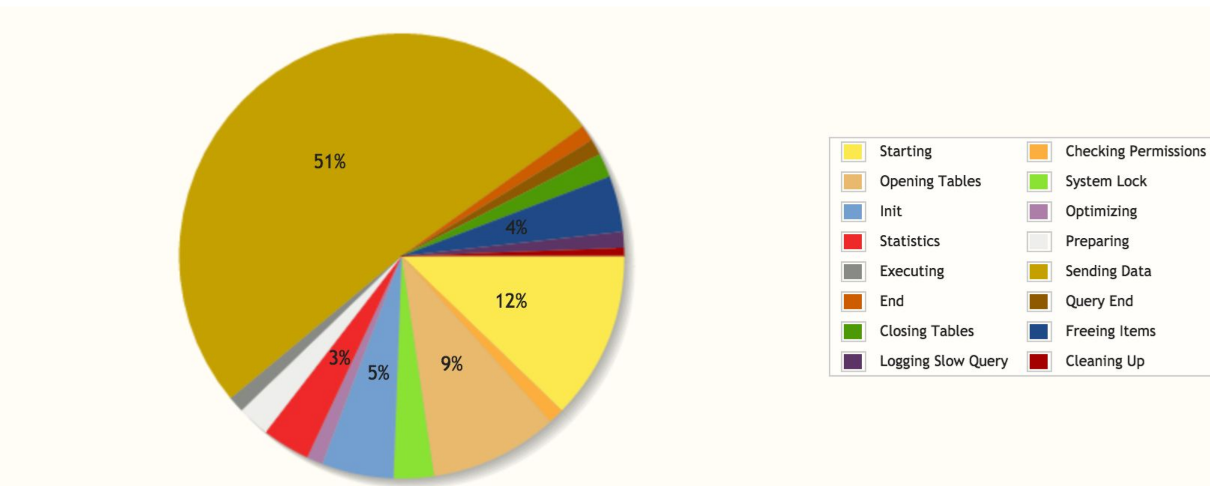
- EXPLAIN SELECT * FROM EMP;

Detailed profile

Order	State	Time
1	Executing	1 μs
2	Sending Data	21 μs
3	Executing	2 μs
4	Sending Data	21 μs
5	Executing	1 μs
6	Sending Data	35 μs
7	Executing	1 μs
8	Sending Data	33 μs
9	Executing	1 μs
10	Sending Data	32 μs

Summary by state

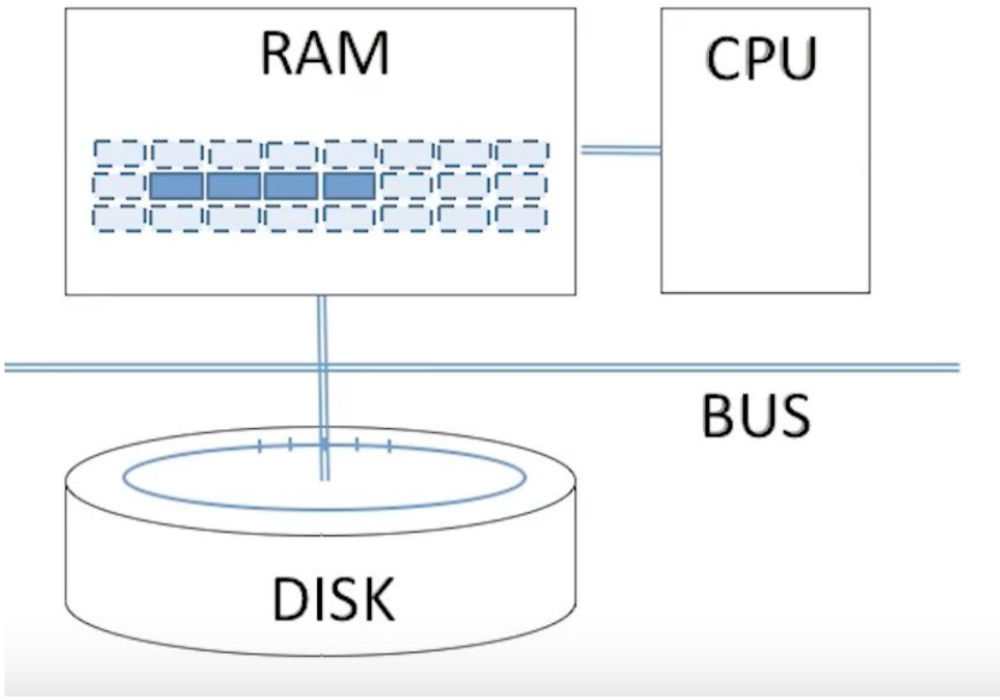
State	Total Time	% Time	Calls	ø Time
Freeing Items	29 μs	1.79%	1	29 μs
Sending Data	21 μs	1.30%	1	21 μs
End	15 μs	0.93%	1	15 μs
Closing Tables	7 μs	0.43%	1	7 μs
Query End	3 μs	0.19%	1	3 μs
Cleaning Up	3 μs	0.19%	1	3 μs
Executing	1 μs	0.06%	1	1 μs
Logging Slow Query	1 μs	0.06%	1	1 μs



Computer Architecture (Overview)

- Access Times:
  - RAM: 30nanosec (3*10^-7)
  - Disk: 10ms (10^-2) for HDD or 0.1ms for SSD
- Cost Computation
  - Only I/O (Disk)
  - CPU costs ignored

=>Tune Index Memory



Choose wisely column types

Choose wisely column types

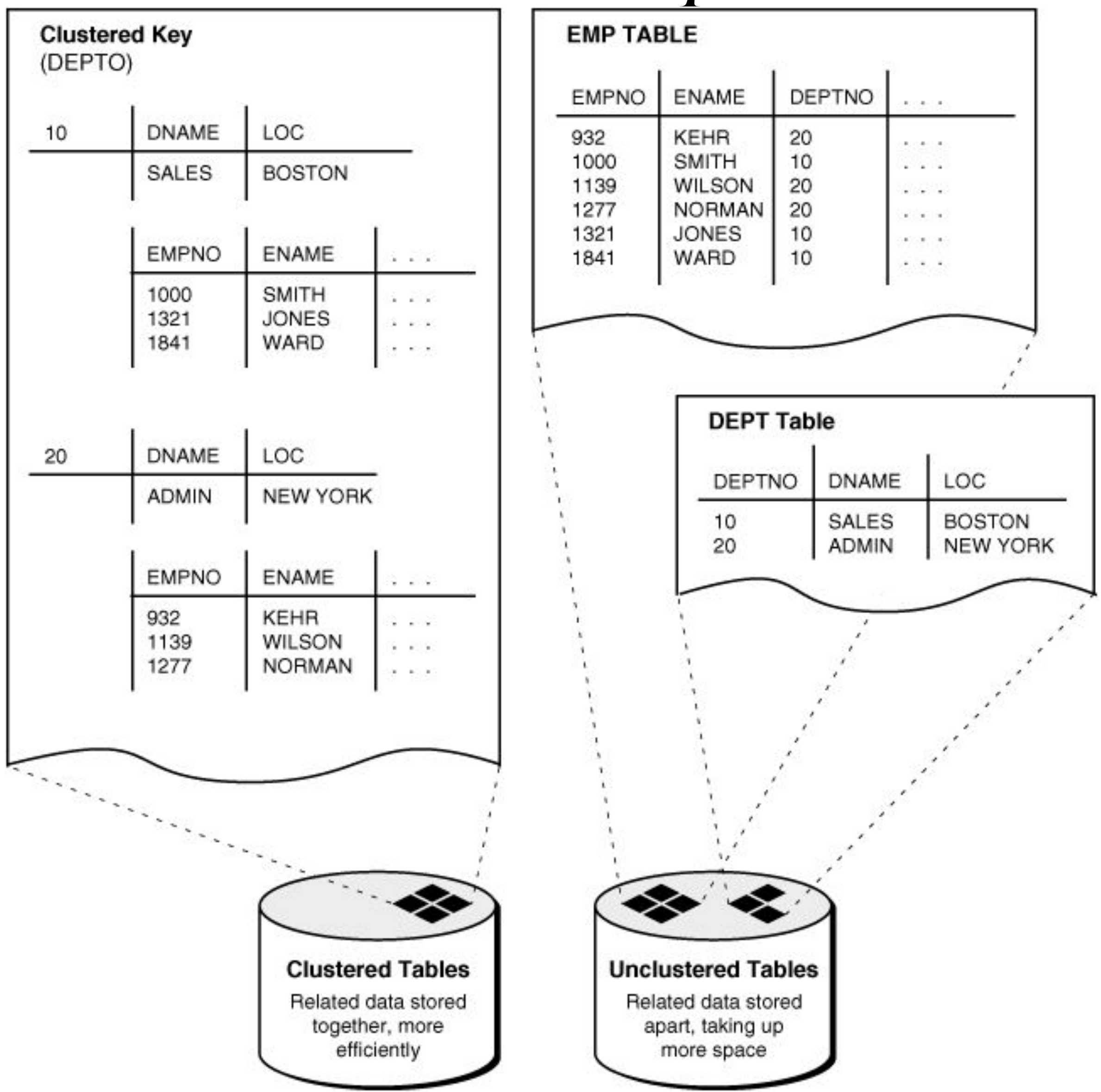
- Don't add Foreign key on Long Text field. Prefer on Integer.
- Don't overuse Long text field when possible

Performances Optimization
Cluster Tables

Cluster Tables

- A cluster provides an optional method of storing table data. A cluster is made up of a group of tables that share the same data blocks. The tables are grouped together because they share common columns and are often used together.
  - For example, the emp and dept table share the deptno column. When you cluster the emp and dept tables (see Figure 22-1), Oracle Database physically stores all rows for each department from both the emp and dept tables in the same data blocks.

Cluster Tables (Example)



Cluster Tables (Benefits)

- Disk I/O is reduced and access time improves for joins of clustered tables. **Faster JOIN**.
- cluster key is the column, or group of columns, that the clustered tables have in common. **Less Storage.**

Cluster Tables (how to choose Tables)

- Use clusters for tables for which the following conditions are true:
  - to The tables are primarily queried--that is, tables that are not predominantly inserted into or updated.
  - Records from the tables are frequently queried

    together or joined.

Cluster (how to choose columns)

- In general, the characteristics that indicate a good cluster index are the same as those for any index.
- A good cluster key has enough unique values so that the group of rows corresponding to each key value fills approximately one data block.
- A cluster index cannot be unique or include a column defined as long.

Cluster (CREATE CLUSTER)

- CREATE CLUSTER emp_dept (deptno NUMBER(3))
  SIZE 600
  TABLESPACE users
  STORAGE (
  INITIAL 200K -- Size of the first extent.
  NEXT 300K -- Size of the next extent.
  MINEXTENTS 2 -- minimum number of extent.
  PCTINCREASE 33); -- percentage by which size of extent grow.

- A cluster index must be created before any rows can be inserted into any clustered table:

```
CREATE INDEX emp_dept_index
    ON CLUSTER emp_dept
    TABLESPACE users
    STORAGE (INITIAL 50K);
```

Cluster (Usage)

- Once you have created the cluster and its cluster indice, you can use it on the tables:

```
CREATE TABLE emp (
    empno NUMBER(5) PRIMARY KEY,
    ename VARCHAR2(15) NOT NULL,
    . . .
    deptno NUMBER(3) REFERENCES dept)
    CLUSTER emp_dept (deptno);

CREATE TABLE dept (
    deptno NUMBER(3) PRIMARY KEY, . . . )
    CLUSTER emp_dept (deptno);
```

Performances Optimization
Caching

Cache (as an Optimizer Hint)

- A result cache is an area of memory, either in the Shared Global Area (SGA) or client application memory, that stores the results of a database query or query block for reuse. The cached rows are shared across SQL statements and sessions unless they become stale.

```
SELECT /*+ RESULT_CACHE */ department_id, AVG(salary)
  FROM hr.employees
 GROUP BY department_id;
```

- Not so much recommended. Better to treat Cache in server side (with fast distributed key-value store such as Redis) or with a your proxy (example: Nginx).

Precise Columns name

*Avoid SELECT **

- You should precise columns name:
  - Increased network traffic
  - Increased CPU usage on client side
  - Some query plan optimizations not possible
  - Server-side memory usage
  - Increased CPU usage on server side

https://tanelpoder.com/posts/reasons-why-select-star-is-bad-for-sql-performance/

- Note: With Column oriented Database, it's extremely more important for performances.

Performances Optimization
Aggregations
(example: with datamart)

Vocabulary OLAP & OLTP & data warehouse & datamart

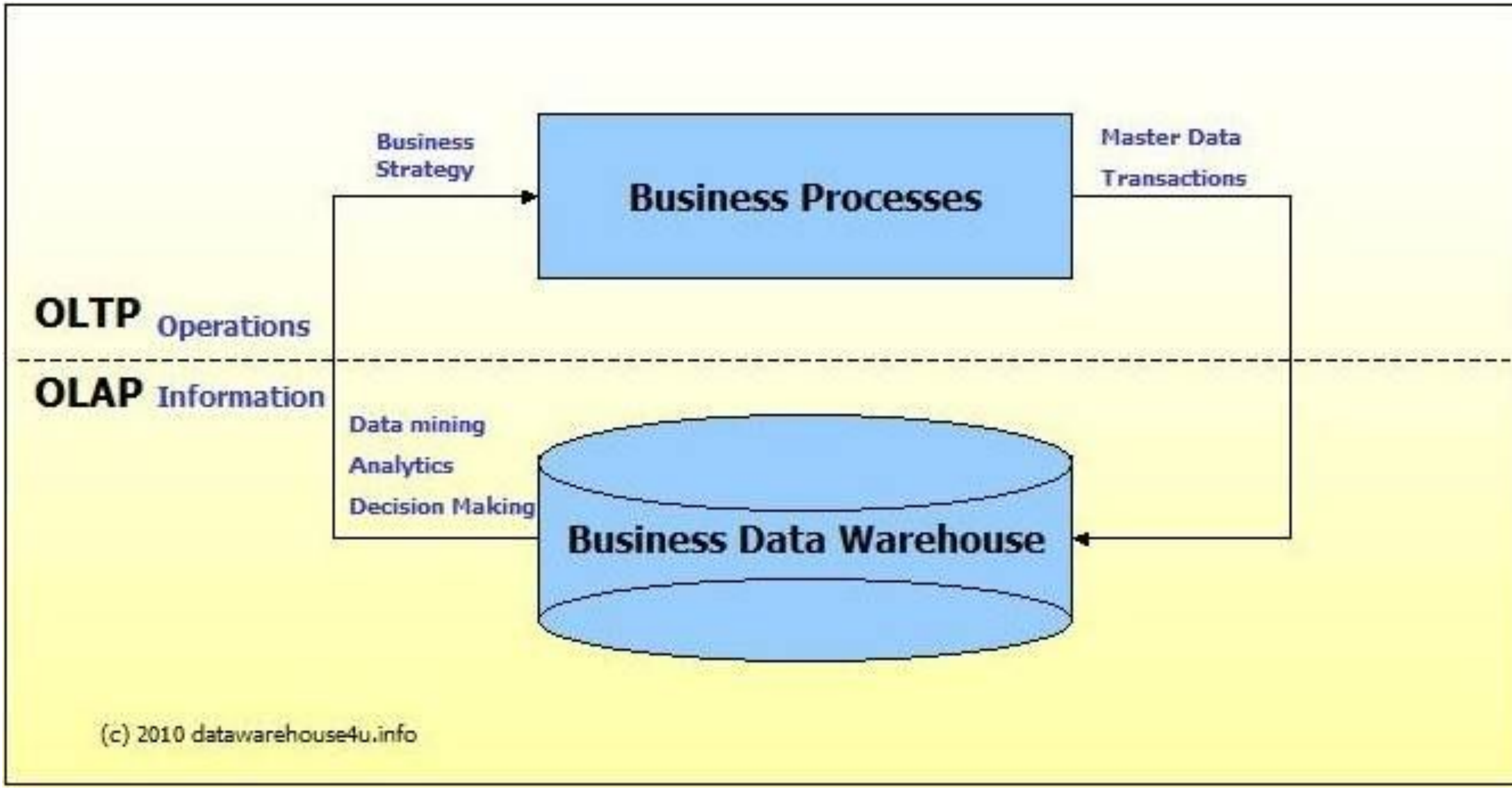
Vocabulary (OLTP)

- OLTP: (On-line Transaction Processing) is characterized by a large number of short on-line transactions (INSERT, UPDATE, DELETE). The main emphasis for OLTP systems is put on very fast query processing, maintaining data integrity in multi-access environments and an effectiveness measured by number of transactions per second. In OLTP database there is detailed and current data, and schema used to store transactional databases is the entity model (usually 3NF).

Vocabulary (OLAP & data warehouse)

- OLAP: (On-line Analytical Processing) is characterized by relatively low volume of transactions. Queries are often very complex and involve aggregations. For OLAP systems a response time is an effectiveness measure. OLAP applications are widely used by Data Mining techniques. In OLAP database there is aggregated, historical data, stored in multi-dimensional schemas (usually star schema).
- Dataware house: is a system used for reporting and data analysis. DWs are central repositories of integrated data from one or more disparate sources.
  - Dataware house uses OLAP operations.

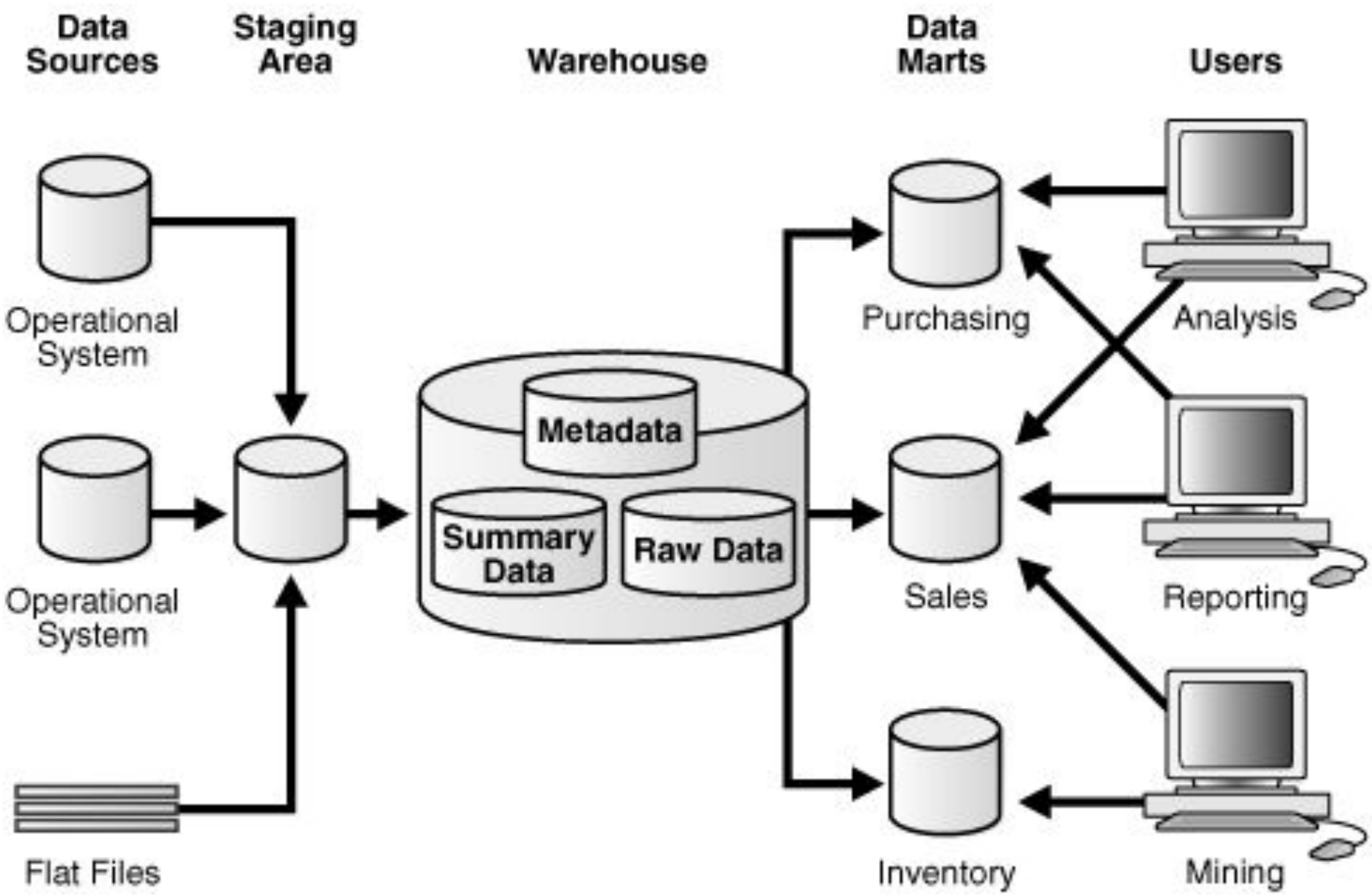
Vocabulary (OLAP vs OLTP)



Vocabulary (data mart)

- A data mart is the access layer of the data warehouse environment that is used to get data out to the users. The data mart is a subset of the data warehouse that is usually oriented to a specific business line or team. Data marts are small slices of the data warehouse.

Vocabulary (datawarehouse to datamart)



Performances Optimization
Partitions

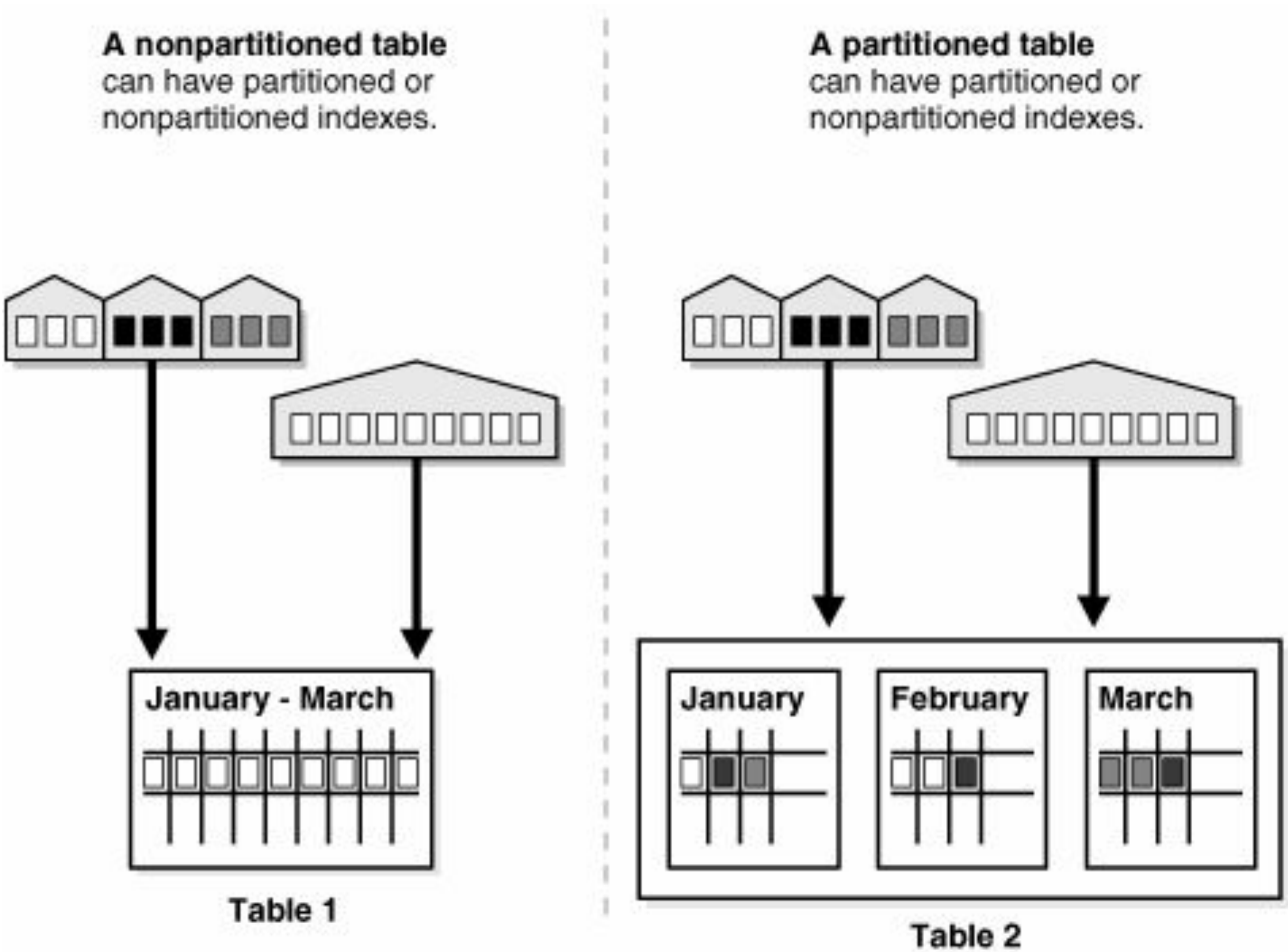
Partitioning (Definition)

- Partitioning addresses key issues in supporting very large tables and indexes by letting you decompose them into smaller and more manageable pieces called partitions.
- Each partition of a table or index must have the same logical attributes, such as column names, datatypes, and constraints, but each partition can have separate physical attributes such as pctfree, pctused, and tablespaces.

Partitioning (Usage)

- Partitioning is useful for many different types of applications, particularly applications that manage large volumes of data.
- OLTP systems often benefit from improvements in manageability and availability.
- Data warehousing systems benefit from performance and manageability.

Partitioning (vs non partitioned)



Partitioning (Partition Key)

- Each row in a partitioned table is unambiguously assigned to a single partition. The partitioning key is comprised of one or more columns that determine the partition where each row will be stored. Oracle automatically directs insert, update, and delete operations to the appropriate partition through the use of the partitioning key.

Partitioning (When ?)

- Tables greater than 2 GB should always be considered as candidates for partitioning.
- Tables containing historical data, in which new data is added into the newest partition. A typical example is a historical table where only the current month's data is updatable and the other 11 months are read only.
- When the contents of a table need to be distributed across different types of storage devices.

Partitioning (Examples)

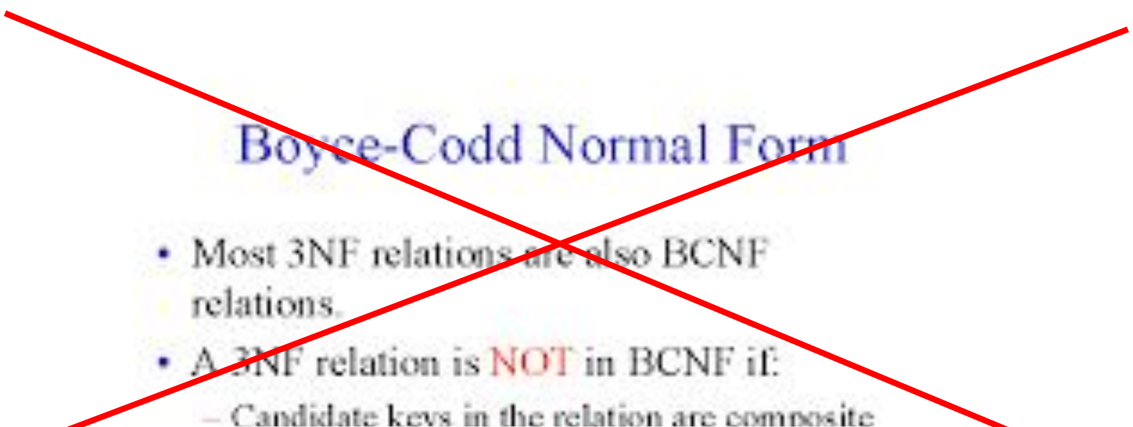
- Range Partitioning Example

```
CREATE TABLE sales_range
(salesman_id  NUMBER(5),
salesman_name VARCHAR2(30),
sales_amount  NUMBER(10),
sales_date    DATE)
PARTITION BY RANGE(sales_date)
(PARTITION sales_jan2000 VALUES LESS THAN(TO_DATE('02/01/2000','MM/DD/YYYY')),
PARTITION sales_feb2000 VALUES LESS THAN(TO_DATE('03/01/2000','MM/DD/YYYY')),
PARTITION sales_mar2000 VALUES LESS THAN(TO_DATE('04/01/2000','MM/DD/YYYY')),
PARTITION sales_apr2000 VALUES LESS THAN(TO_DATE('05/01/2000','MM/DD/YYYY')));
```

- List Partitioning Example

```
CREATE TABLE sales_list
(salesman_id  NUMBER(5),
salesman_name VARCHAR2(30),
sales_state   VARCHAR2(20),
sales_amount  NUMBER(10))
PARTITION BY LIST(sales_state)
(PARTITION sales_west VALUES('California', 'Hawaii'),
PARTITION sales_east VALUES ('New York', 'Virginia', 'Florida'),
PARTITION sales_central VALUES('Texas', 'Illinois'),
PARTITION sales_other VALUES(DEFAULT));
```

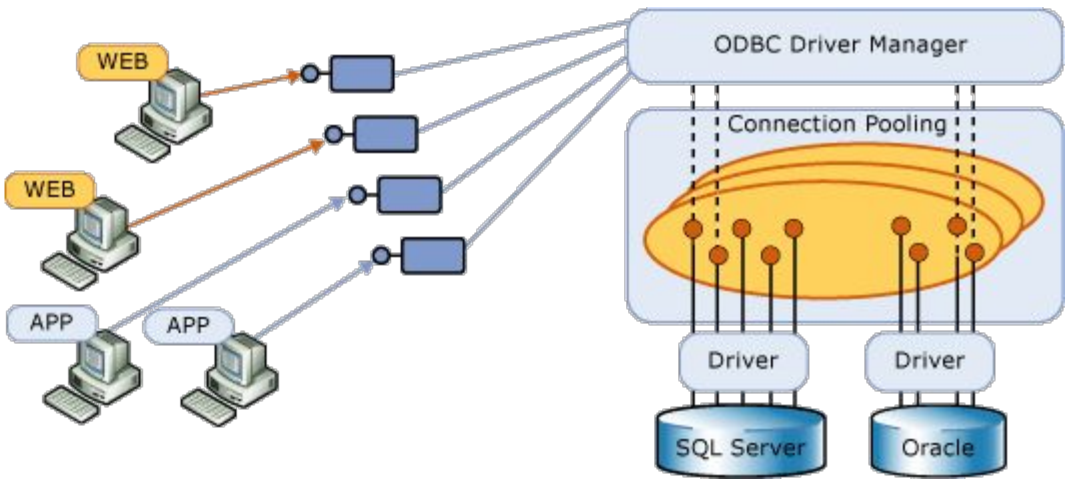
Performances Optimization
Denormalization & Others
(to be continued in Big Data module)



Performances Optimization Pool

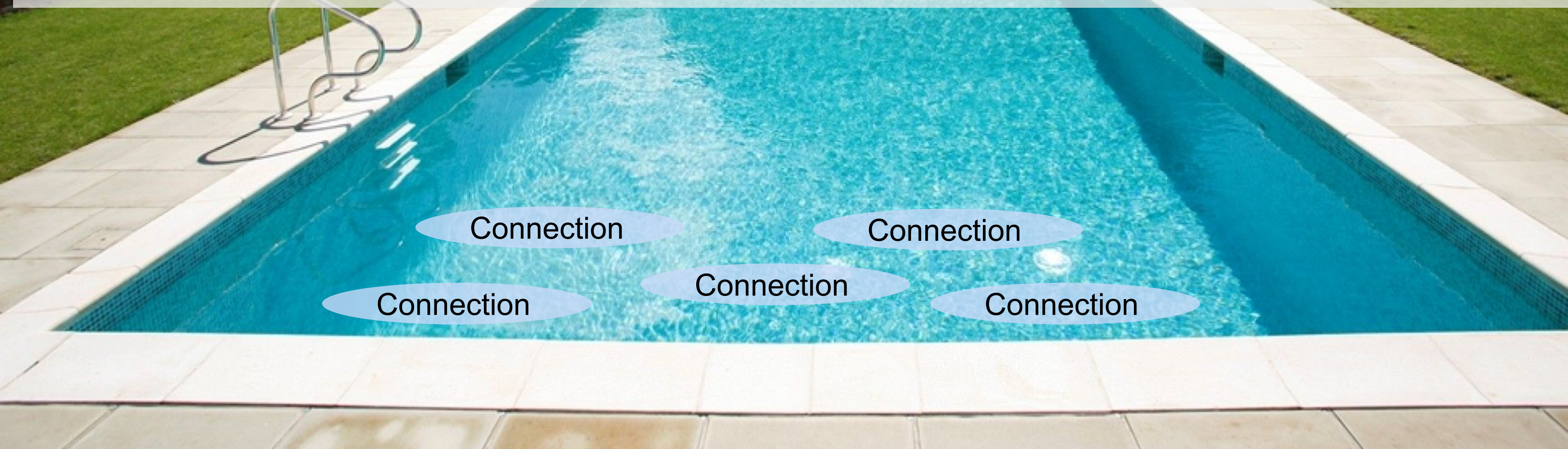
Connection Pool (Definition)

- A connection pool contains a group of JDBC connections that are created when the connection pool is registered.
- Connection pools use a JDBC driver to create physical database connections.
- Your application borrows a connection from the pool, uses it, then returns it to the pool by closing it.



Connection Pool (Reasons)

- First, the run time creation of new software objects is often more expensive in terms of performance and memory than the reuse of previously created objects.
- Second, garbage collection is an expensive process so when we reduce the number of objects to clean up we generally reduce the garbage collection load.



Naming Conventions

Naming Conventions Best Practices

PascalCase, camelCase, snake_case, SCREAMING_CASE ?

A proposal is the lower snake case:

- table_name (employee_company)
- field_name (first_name)

Why ?

- Some database only allow lowercase. Then to differentiate words, it’s better to have an underscore to separate them.

Plural or Singular ?

Majority agree on **singular**.

Debate is long: https://stackoverflow.com/a/5841297

- Convenience
- Aesthetic and Order
- Simplicity
- Globalization
- Use uninflected noun

Simples rules

- Use English
- Don’t use special chars unless underscore
- Be consistent
- Limit the use of abbreviations / acronym (can be hard to understand)
- Make the name readable
- Avoid too long names
- Avoid number (it sounds generally weird / bad design)

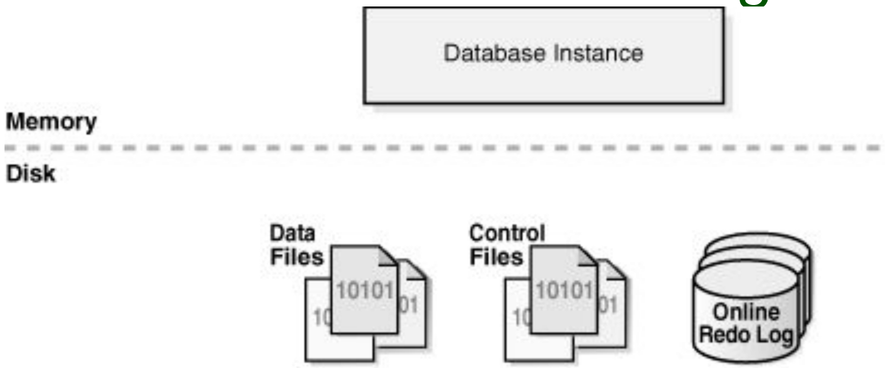
https://www.isbe.net/documents/sql_server_standards.pdf

Organizing
Physical Storage Structures

Physical Storage Structures

An Oracle database is a set of files that store Oracle data in persistent disk storage:

- ❖ **Data files and temp files**: A data file is a physical file on disk that was created by Oracle Database and contains data structures such as tables and indexes.
- ❖ A **control file** is a root file that tracks the physical components of the database.
- ❖ **The online redo log files** is a set of files containing records of changes made to data.



Mechanisms for Storing Database Files

Several mechanisms are available for allocating and managing the storage of these files. The most common mechanisms include:

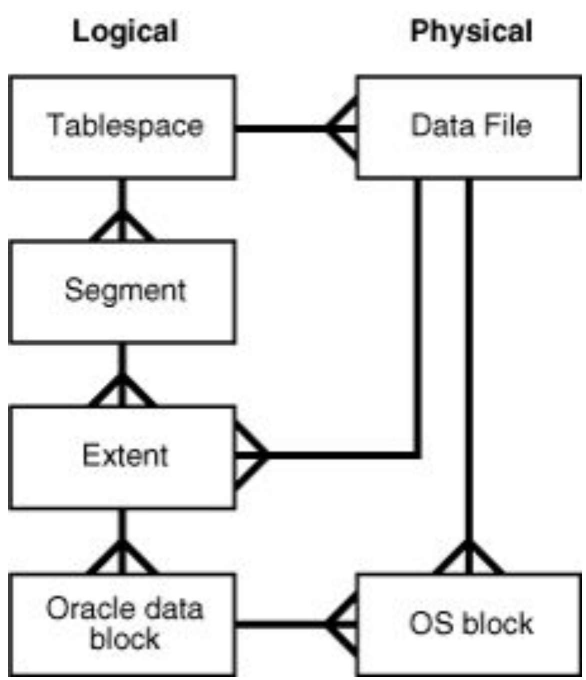
- ❖ Oracle Automatic Storage Management (Oracle ASM)
- ❖ Operating system file system
- ❖ Raw devices
- ❖ Cluster file system

Organizing

Logical Structures

Overview of Logical Storage

- Oracle Database allocates logical space for all data in the database. The logical units of database space allocation are data blocks, extents, segments, and tablespaces.

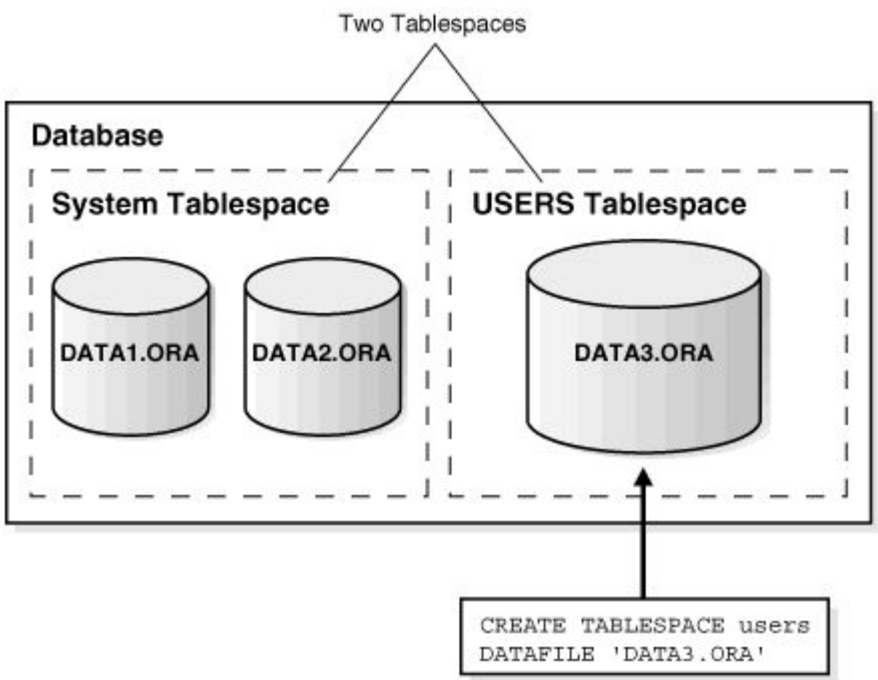


Organizing

Tablespaces, Data Blocks, Extents, and Segments

Tablespaces

- Oracle stores data logically in tablespaces and physically in datafiles associated with the corresponding tablespace.

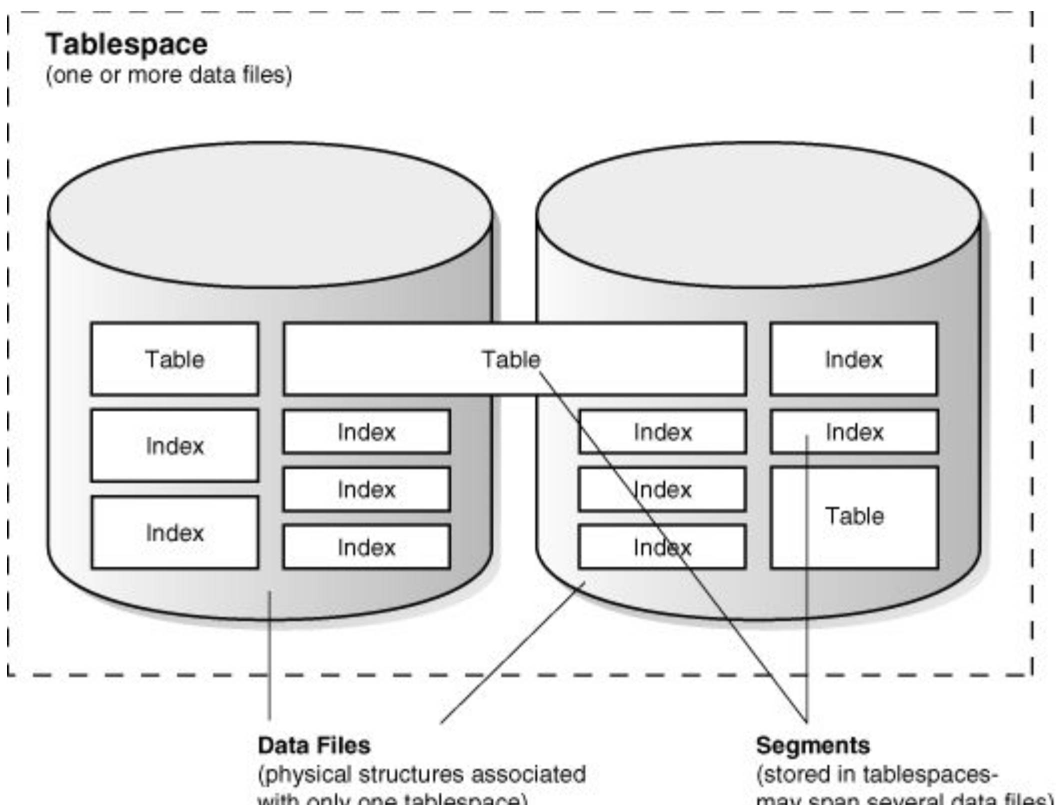


Tablespaces

- A tablespace is a database storage unit that groups related logical structures together.
- Using multiple tablespaces permits: to
  - Separate user data from data dictionary data to reduce I/O contention.
  - Separate data of one application from the data of another to prevent multiple applications from being affected if a tablespace must be taken offline.
  - Store the data files of different tablespaces on different disk drives to reduce I/O contention.
  - Take individual tablespaces offline while others remain online, providing better overall availability.
  - Optimizing tablespace use by reserving a tablespace for a particular type of database use, such as high update activity, read-only activity, or temporary segment storage.
  - Back up individual tablespaces.

Tablespaces and Data Files

- At the operating system level, Oracle Database stores database data in data files.
- Each tablespace consists of one or more data files.
- A segment can span 1 or more data files, but can't span multiple tablespace.

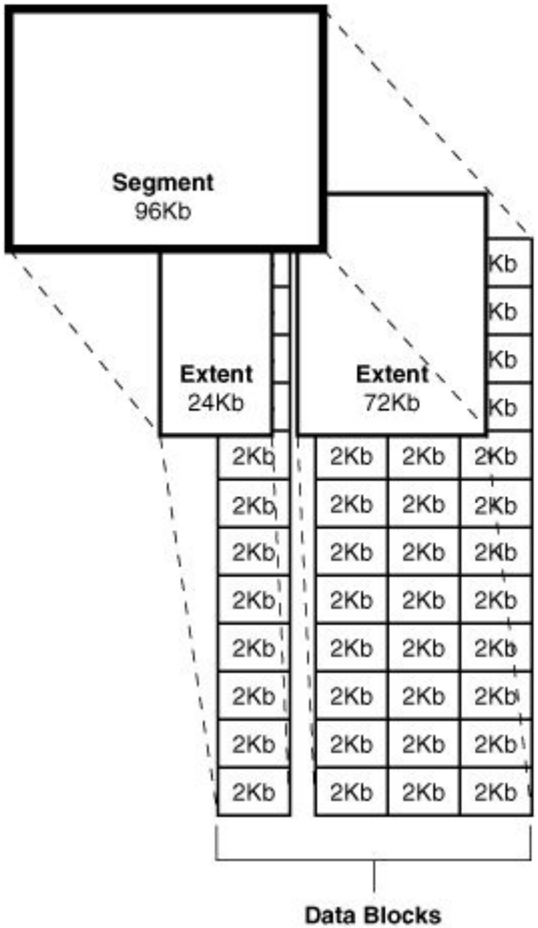


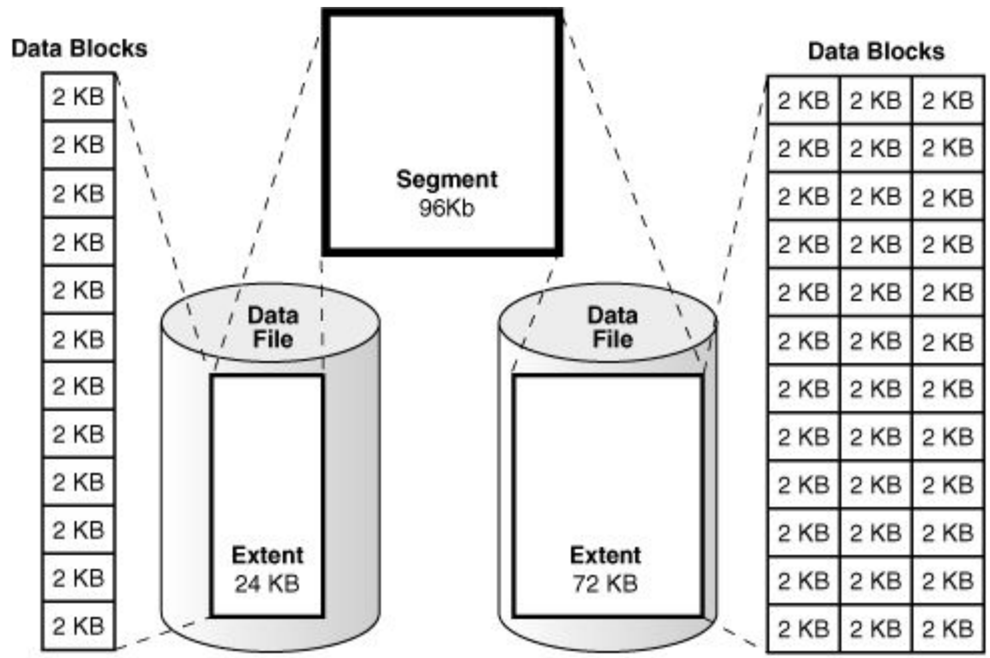
Tablespaces (usage example)

- CREATE TABLESPACE myTableSpace LOCATION '/mnt/sda1/oracle/data';
- CREATE TABLE foo(i int) TABLESPACE space1;
- SELECT spcname FROM pg_tablespace; // List all tablespaces
- CREATE USER user123 IDENTIFIED BY password DEFAULT TABLESPACE myTableSpace TEMPORARY TABLESPACE TEMP QUOTA 50 M ON myTableSpace ACCOUNT UNLOCK;
- GRANT USE OF TABLESPACE myTableSpace TO user123;

Data Blocks, Extents, and Segments

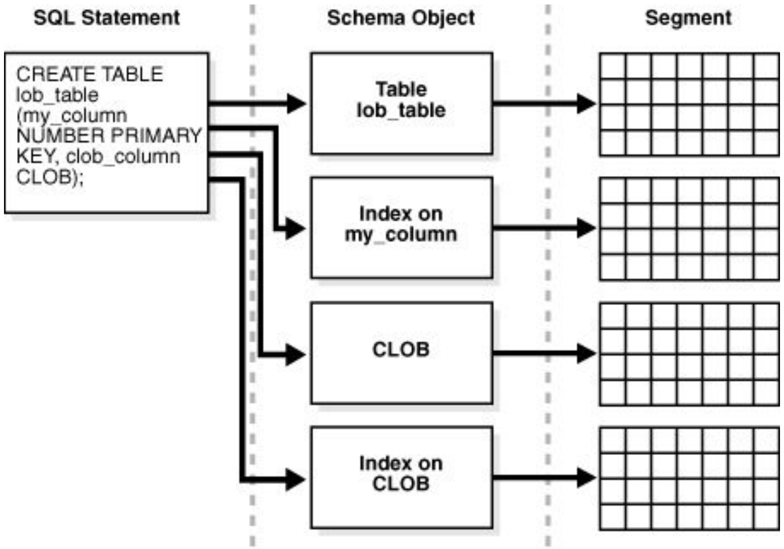
❖ Oracle allocates logical database space for all data in a database. The units of database space allocation are data blocks, extents, and segments.





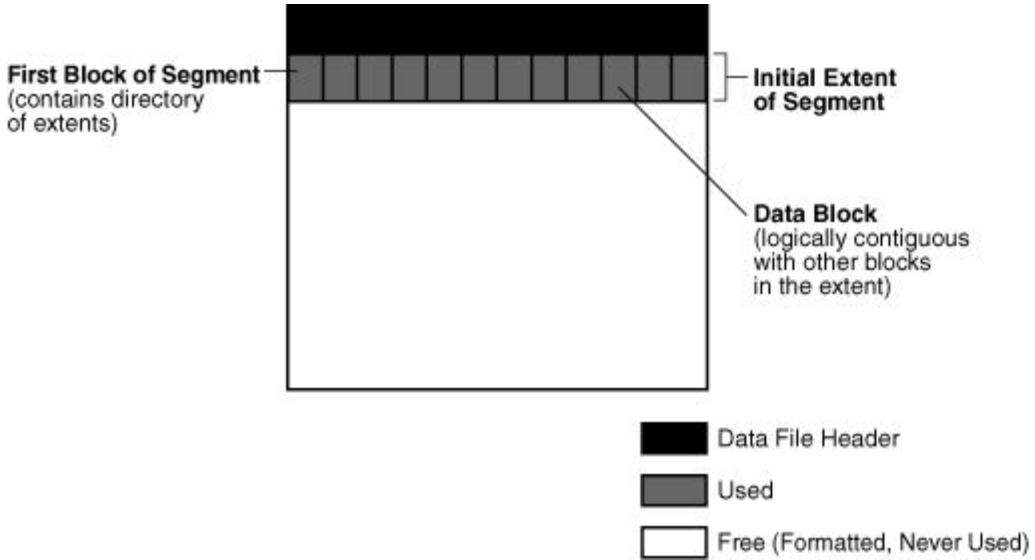
Segments

- A segment is a set of extents that contains all the data for a logical storage structure within a tablespace. For example, Oracle Database allocates one or more extents to form the data segment for a table. The database also allocates one or more extents to form the index segment for a table.



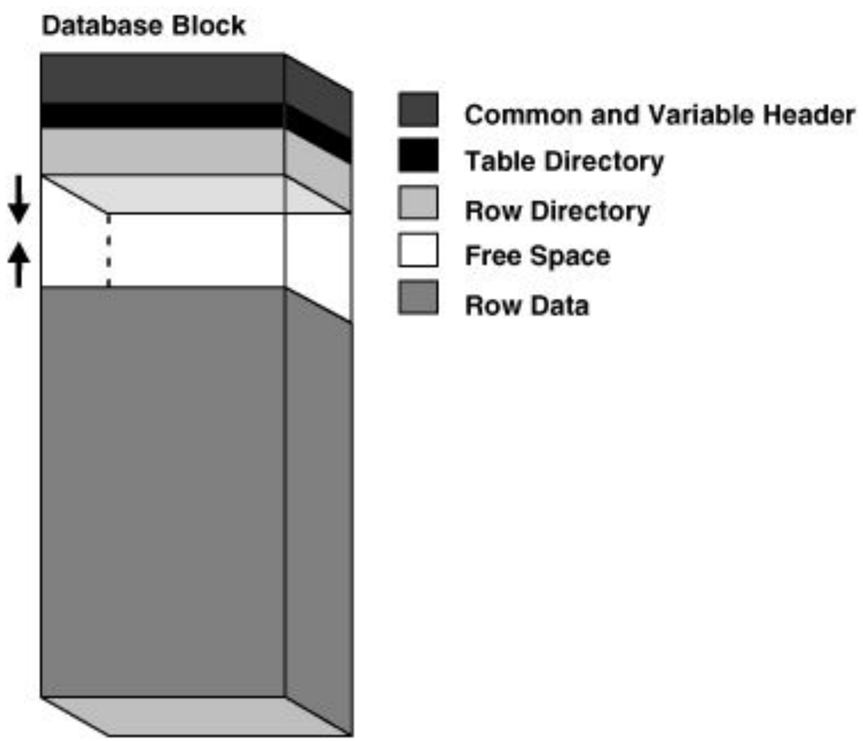
Extents

- An extent is a logical unit of database storage space allocation made up of contiguous data blocks. Data blocks in an extent are logically contiguous but can be physically spread out on disk because of RAID striping and file system implementations.



Data Blocks

- A **data block** is the smallest unit of data used by a database. Each operating system has a block size (DB_BLOCK_SIZE).
- Oracle Database manages the logical storage space in the data files of a database in units called data blocks, also called Oracle blocks or pages. A data block is the minimum unit of database I/O.

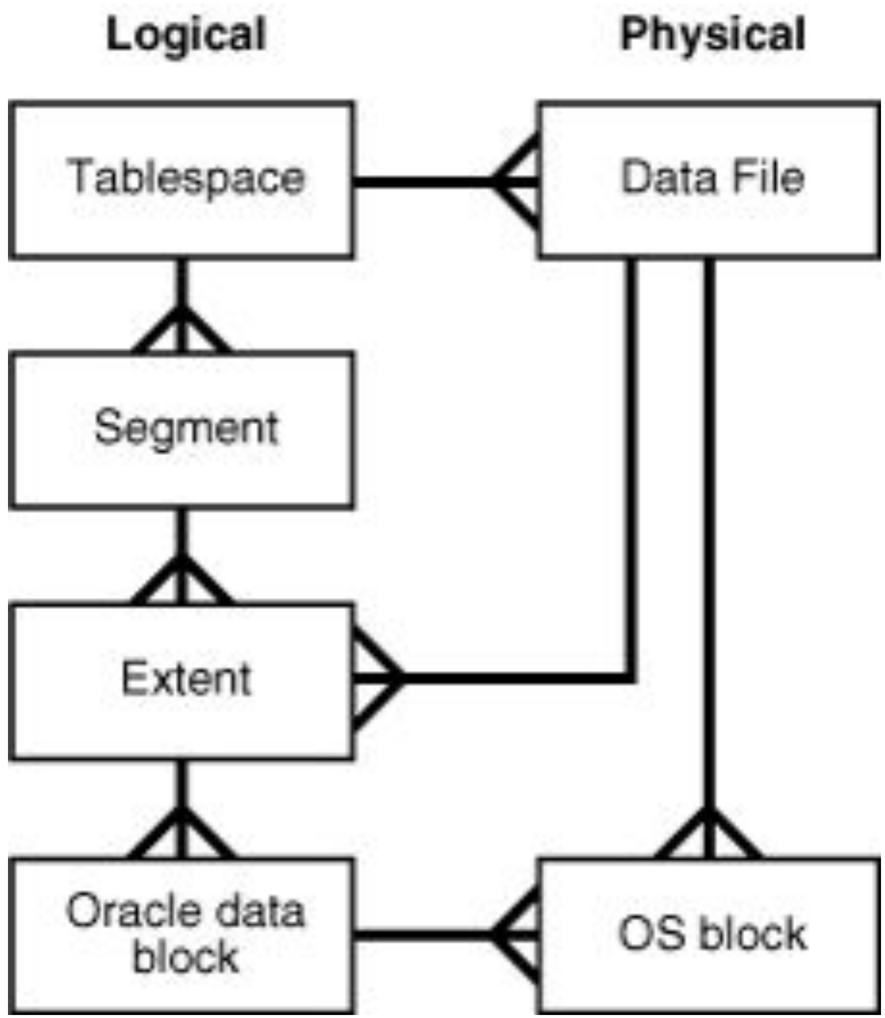


Storage Clause

- The storage_clause lets you specify how Oracle Database should store a database object. Storage parameters affect both how long it takes to access data stored in the database and how efficiently space in the database is used.
- Example:

```
CREATE TABLE divisions
   (div_no     NUMBER(2),
    div_name   VARCHAR2(14),
    location   VARCHAR2(13) )
    STORAGE  (
              INITIAL 100K
              NEXT     50K
              MINEXTENTS 1
              MAXEXTENTS 50
              PCTINCREASE 5);
```

Overview


Logical
Physical
Tablespace
Data File
Segment
Extent
Oracle data block
OS block