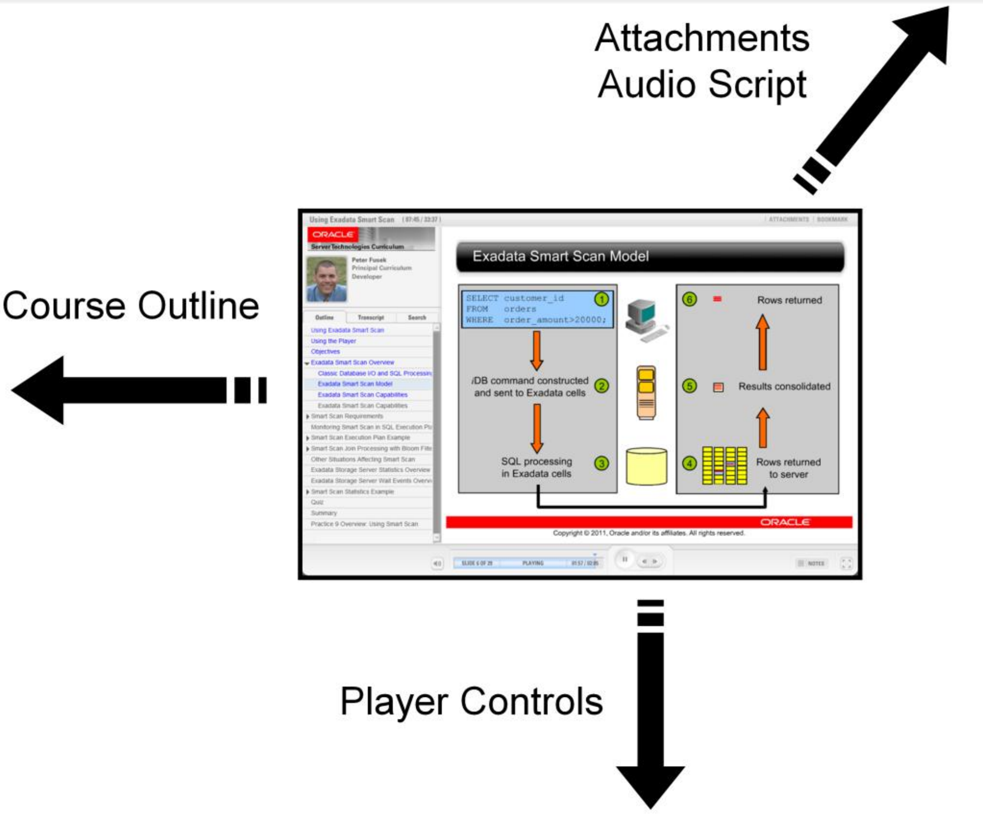
Using Exadata Smart Scan

ORACLE

Hello, and welcome to this online, self-paced course entitled Using Exadata Smart Scan. My name is Peter Fusek. I am a curriculum developer at Oracle, and in various roles I have helped customers to get the most out of Oracle products since 1992. Today, I will be you tour guide for this course.

Using the Player


Attachments
Audio Script
Course Outline
Player Controls
Using Exadata Smart Scan (07:45 / 33:37)
ATTACHMENTS | BOOKMARK
ORACLE
Server Technologies Curriculum
Peter Fusek
Principal Curriculum Developer
Outline
Transcript
Search
Using Exadata Smart Scan
Using the Player
Objectives
Exadata Smart Scan Overview
Classic Database I/O and SQL Processing
Exadata Smart Scan Model
Exadata Smart Scan Capabilities
Exadata Smart Scan Capabilities
Smart Scan Requirements
Monitoring Smart Scan in SQL Execution Plans
Smart Scan Execution Plan Example
Smart Scan Join Processing with Bloom Filters
Other Situations Affecting Smart Scan
Exadata Storage Server Statistics Overview
Exadata Storage Server Wait Events Overview
Smart Scan Statistics Example
Quiz
Summary
Practice 9 Overview: Using Smart Scan
Exadata Smart Scan Model
1
iDB command constructed and sent to Exadata cells
2
SQL processing in Exadata cells
3
4
Rows returned to server
5
Results consolidated
6
Rows returned
ORACLE
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.
SLIDE 6 OF 29
PLAYING
01:57 / 02:05
NOTES

```
SELECT customer_id
FROM   orders
WHERE  order_amount>20000;
```

Before we begin, now might be a good time to take a look at some of the features of this Flash-based course player. Feel free to skip this slide and start the lecture if you've attended similar Oracle online courses in the past.

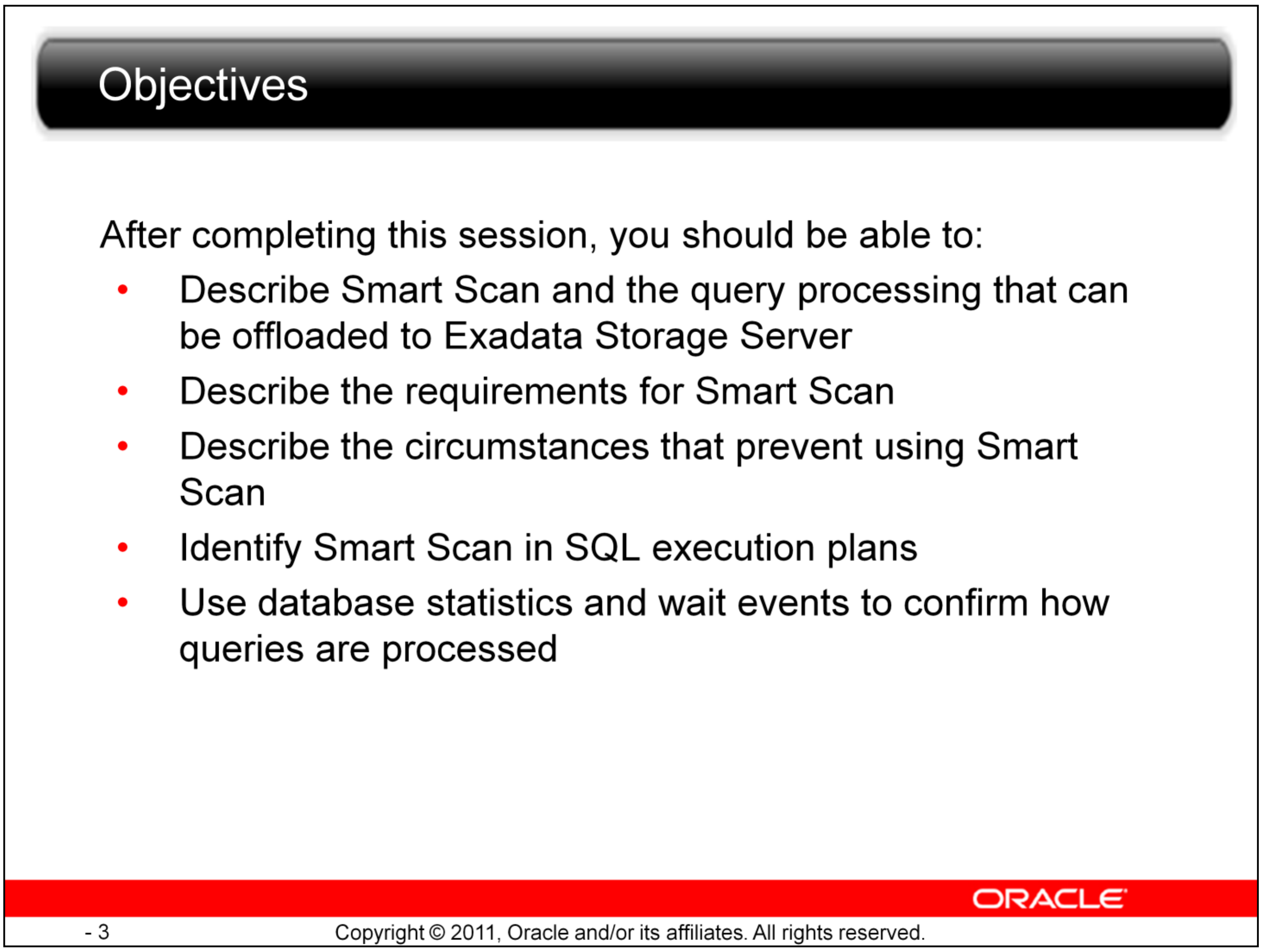
To your left, you will find a hierarchical course outline. This course enables and even encourages you to go at your own pace, which means you are free to skip over topics you already feel confident on, or jump right to a feature that really interests you, or go back and review topics that were already covered. Simply click on a course section to expand its contents and then select an individual slide. However, note that by default we will automatically walk you through the entire course without requiring you to use the outline.

Standard Flash player controls are also found at the bottom of the player, including pause, previous, and next buttons. There is also an interactive progress bar to fast forward or rewind the current slide. Interactive slides may have additional controls and buttons along with instructions on how to use them.

Also found at the bottom of the player is a panel containing any additional reference notes for the current slide. Feel free to read these reference notes at the conclusion of the course, in which case you can minimize this panel and restore it later. Or if you prefer you can pause and read them as we go along.

Various handouts may be available from the Attachments button, including the audio narration scripts for this course.

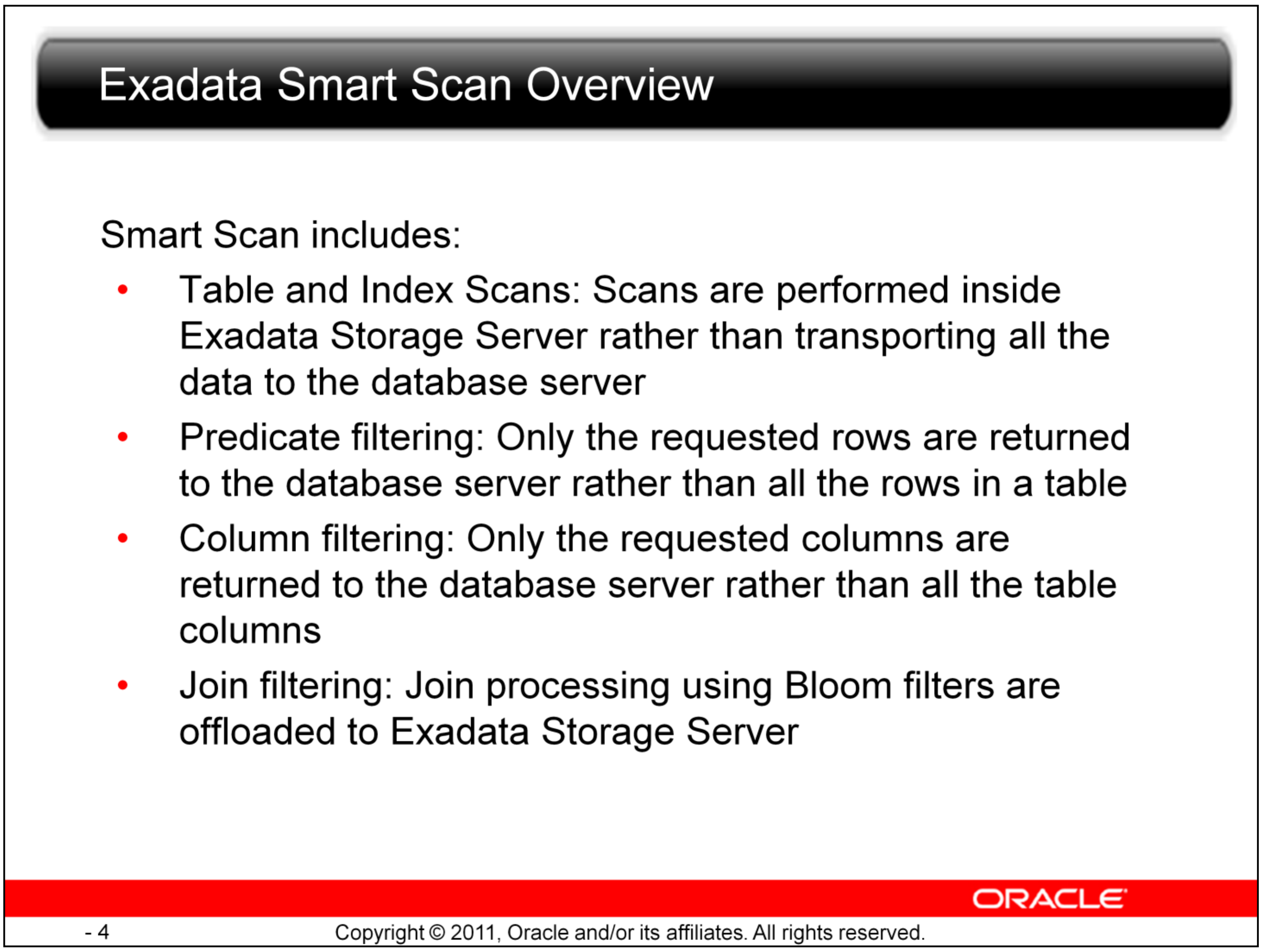
The course will now pause, so feel free to take some time and explore the interface. Then when you're ready to continue, click the next button below or alternatively click the Objectives slide in the course outline.



Objectives

After completing this session, you should be able to:

- Describe Smart Scan and the query processing that can be offloaded to Exadata Storage Server
- Describe the requirements for Smart Scan
- Describe the circumstances that prevent using Smart Scan
- Identify Smart Scan in SQL execution plans
- Use database statistics and wait events to confirm how queries are processed



Exadata Smart Scan Overview

Smart Scan includes:

- Table and Index Scans: Scans are performed inside Exadata Storage Server rather than transporting all the data to the database server
- Predicate filtering: Only the requested rows are returned to the database server rather than all the rows in a table
- Column filtering: Only the requested columns are returned to the database server rather than all the table columns
- Join filtering: Join processing using Bloom filters are offloaded to Exadata Storage Server

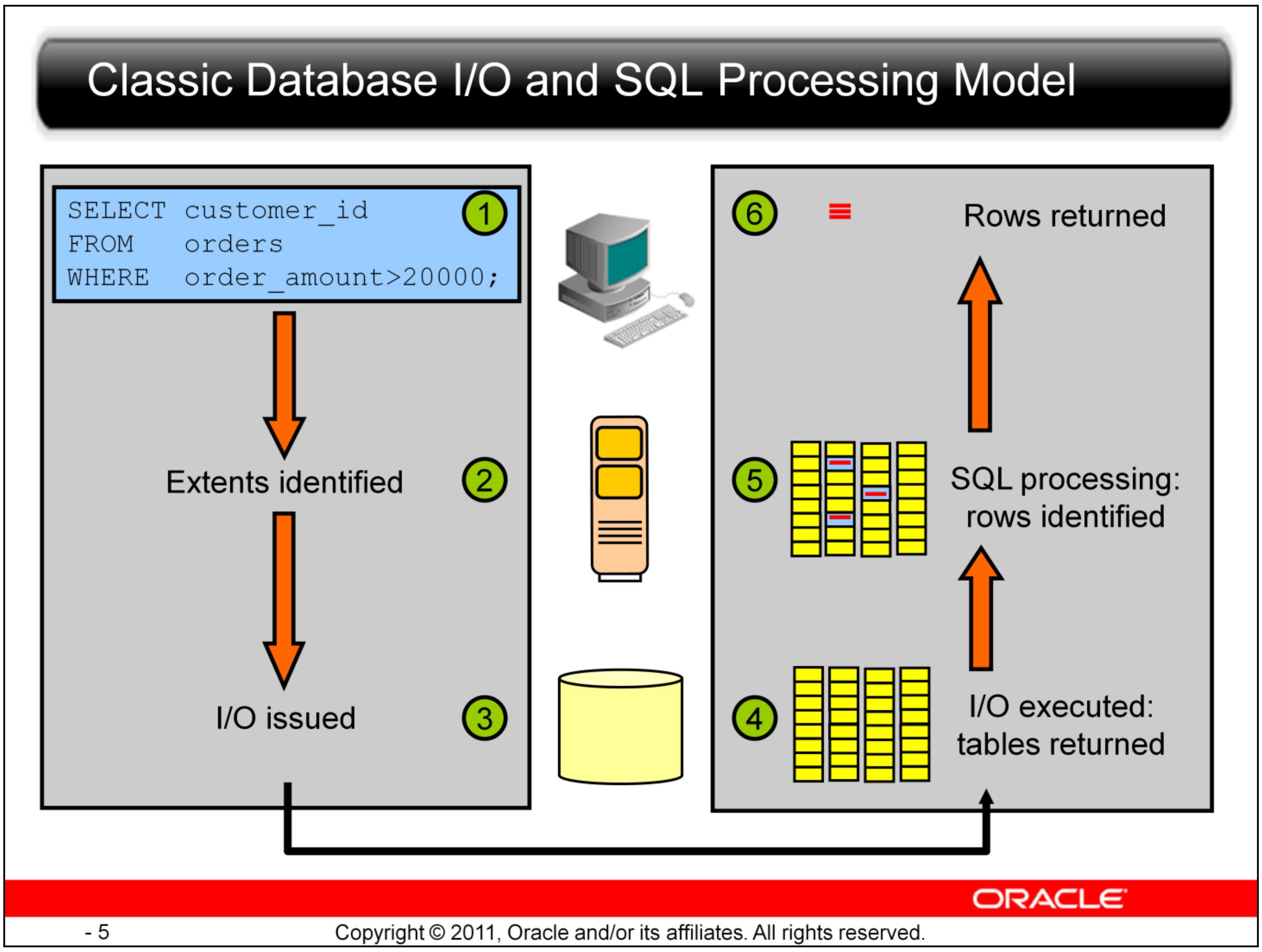
ORACLE

- 4

One of the most powerful features of Database Machine is that data search and retrieval processing can be offloaded to the Exadata Storage Servers. This feature is called Smart Scan. Using this Smart Scan, Oracle Database can optimize the performance of operations that perform table and index scans by performing the scans inside Exadata Storage Server, rather than transporting all the data to the database server. This principle can also be applied to encrypted data and compressed data. In addition, various aspects of SQL processing can be offloaded to the Exadata Storage Server so that it is performed close to the data and so that the amount of data transported back to the database server can be minimized.

The slide lists the key database functions which are integrated with Exadata Storage Server. The term Smart Scan is also used to describe instances when these functions are performed by Exadata Storage Server.

This session focuses on the requirements for performing Smart Scan processing, and how you can use core SQL monitoring capabilities within Oracle Database to identify Smart Scans.



Classic Database I/O and SQL Processing Model

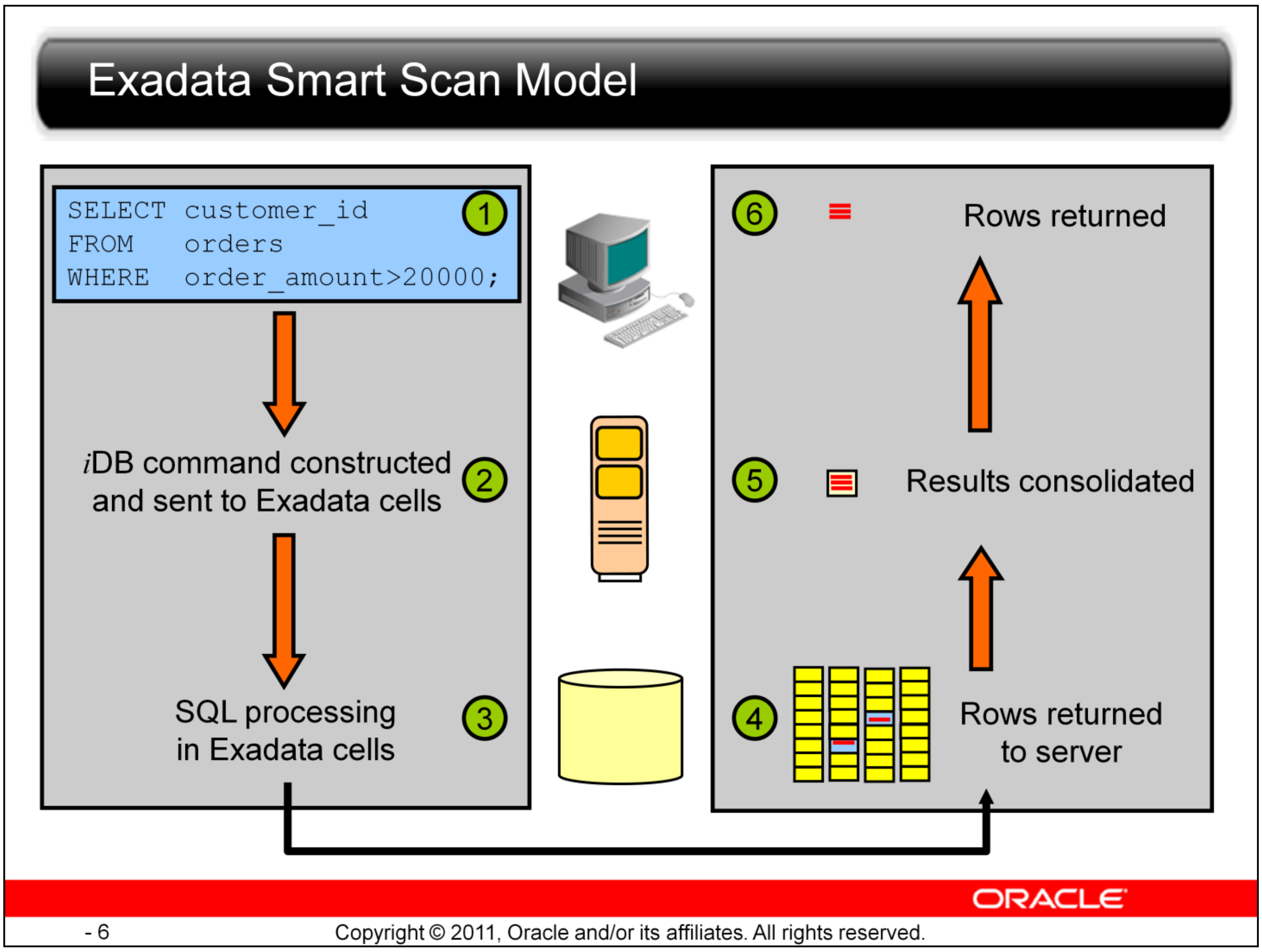
```
SELECT customer_id
FROM   orders
WHERE  order_amount>20000;
```

1
Extents identified
2
I/O issued
3
6
Rows returned
5
SQL processing: rows identified
4
I/O executed: tables returned
ORACLE

To understand how Smart Scan works, let's start by considering how Oracle processes queries using conventional storage. With conventional storage, all the database intelligence resides in the software on the database server. To illustrate how SQL processing is performed in this architecture, an example of a table scan is shown in the graphic on the screen:

1. The client issues a `SELECT` statement with a predicate to filter a table and return only the rows of interest to the user.
2. The database kernel maps this request to the file and extents containing the table.
3. The database kernel issues the I/Os to read all the table blocks.
4. All the blocks for the table being queried are read into memory.
5. SQL processing is conducted against the data blocks searching for the rows that satisfy the predicate.
6. The required rows are returned to the client.

As is often the case with large queries, the predicate filters out most of the rows in the table. Yet all the blocks from the table need to be read, transferred across the storage network, and copied into memory. Many more rows are read into memory than required to complete the requested SQL operation. This generates a large amount of unproductive I/O, which wastefully consumes resources and impacts application throughput and response time.



Exadata Smart Scan Model

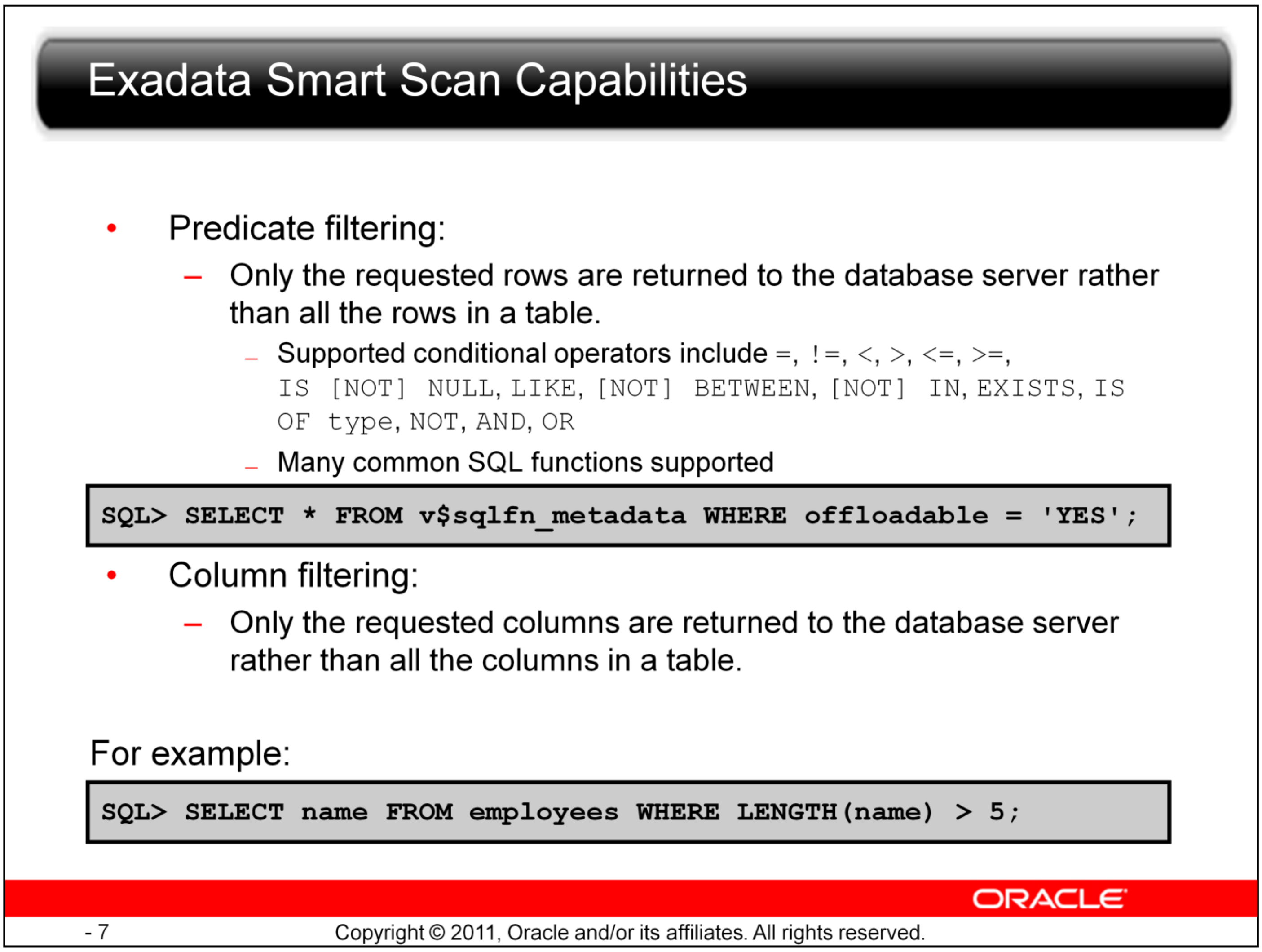
```
SELECT customer_id
FROM   orders
WHERE  order_amount>20000;
```

1
iDB command constructed and sent to Exadata cells
2
SQL processing in Exadata cells
3
6
Rows returned
5
Results consolidated
4
Rows returned to server
ORACLE

On Exadata Database Machine, database operations are handled differently. Queries that perform table scans can be processed within Exadata cells and return only the required subset of data to the database server. Row filtering, column filtering, some join processing, and other functions can be performed within Exadata cells. Exadata Storage Server uses a special direct-read mechanism for Smart Scan processing. The graphic on the screen illustrates how a table scan operates with Exadata cell storage:

1. The client issues a `SELECT` statement to return some rows of interest.
2. The database kernel determines that the data is stored on Exadata cells, so an iDB command representing the SQL command is constructed and sent to the Exadata cells.
3. The Exadata Storage Server software scans the data blocks to extract the relevant rows and columns that satisfy the SQL command.
4. Exadata cells return to the database instance iDB messages containing the requested rows and columns of data. These results are not block images, so they are not stored in the buffer cache.
5. The database kernel consolidates the result sets from across all the Exadata cells. This is similar to how the results from a parallel query operation are consolidated.
6. The rows are returned to the client.

Moving SQL processing off the database server frees server CPU cycles and eliminates a massive amount of unproductive I/O transfers. These resources are free to better service other requests. Queries run faster, and more of them can be processed.



Exadata Smart Scan Capabilities

- Predicate filtering:
  - Only the requested rows are returned to the database server rather than all the rows in a table.
    - Supported conditional operators include =, !=, <, >, <=, >=, IS [NOT] NULL, LIKE, [NOT] BETWEEN, [NOT] IN, EXISTS, IS OF type, NOT, AND, OR
    - Many common SQL functions supported

```
SQL> SELECT * FROM v$sqlfn_metadata WHERE offloadable = 'YES';
```

- Column filtering:
  - Only the requested columns are returned to the database server rather than all the columns in a table.

For example:

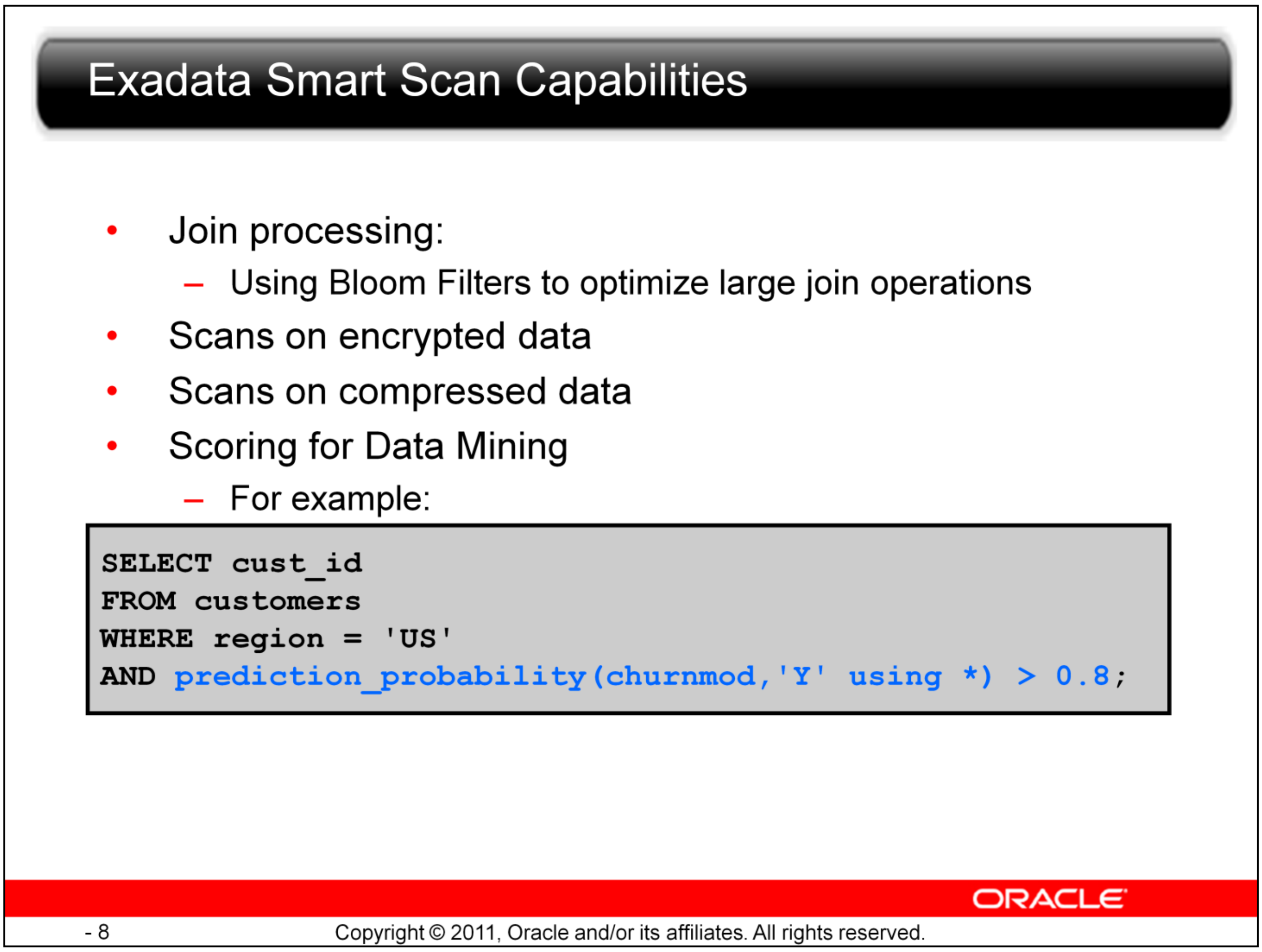
```
SQL> SELECT name FROM employees WHERE LENGTH(name) > 5;
```

- 7

Exadata Storage Server enables predicate filtering for table scans. Rather than returning all the rows for the database to evaluate, Exadata Storage Server returns only the rows that match the filter condition. A list of conditional operators that are supported by predicate filtering is shown on the screen. In addition, many common SQL functions can be evaluated by Exadata Storage Server during predicate filtering. For a full list of functions that can be evaluated by Exadata cell, use the query shown on the screen.

Exadata Storage Server provides column filtering, also called column projection, for table scans. Only the requested columns are returned to the database server, rather than all columns in a table. For tables with many columns, or columns containing LOBs, the I/O bandwidth saved by column filtering can be very large.

When used together, the combination of predicate and column filtering dramatically improves performance and reduces I/O bandwidth consumption. For example, without filtering, the entire employees table must be sent from the storage to the database server to satisfy the query at the bottom of the screen. With predicate and column filtering, only the employee names that are longer than five characters are sent to the database servers.



Exadata Smart Scan Capabilities

- Join processing:
  - Using Bloom Filters to optimize large join operations
- Scans on encrypted data
- Scans on compressed data
- Scoring for Data Mining
  - For example:

```
SELECT cust_id
FROM customers
WHERE region = 'US'
AND prediction_probability(churnmod,'Y' using *) > 0.8;
```

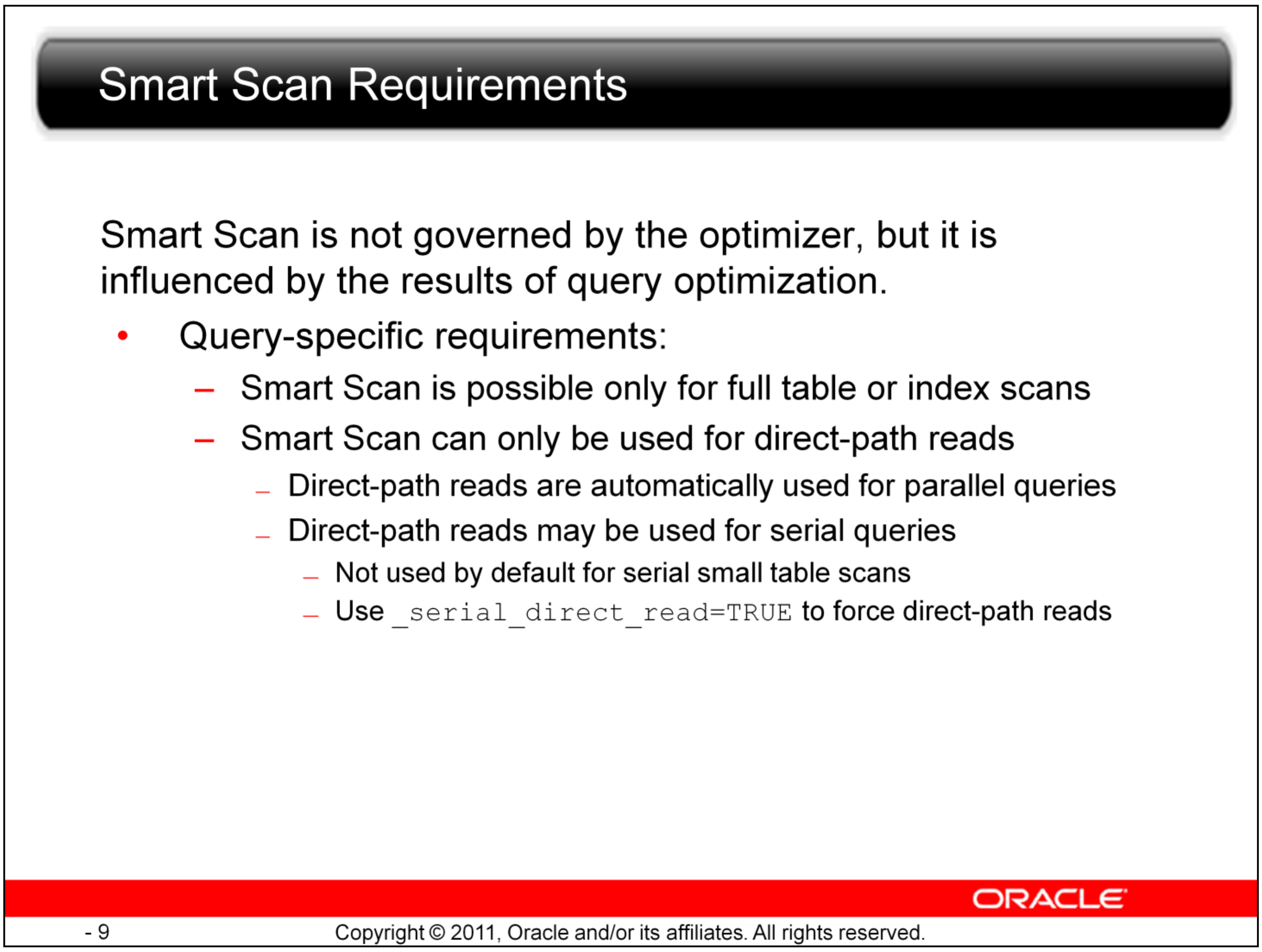
ORACLE

Exadata Storage Server can optimize join processing for large join operations. This is implemented by using a Bloom Filter, which is a very efficient probabilistic method to determine whether an element is a member of a set. We'll talk about this more later in the course.

Exadata Storage Server performs Smart Scans on encrypted tablespaces and encrypted columns. For encrypted tablespaces, Exadata Storage Server can decrypt blocks and return the decrypted blocks to Oracle Database, or it can perform row and column filtering on encrypted data. Significant CPU savings can be made within the database server by offloading the CPU-intensive decryption task to Exadata cells.

Smart Scan works in conjunction with Exadata Hybrid Columnar Compression so that column projection and row filtering can be executed along with decompression at the storage level to save CPU cycles on the database servers.

Exadata Storage Server can perform scoring functions, such as `PREDICTION_PROBABILITY`, for data mining models. This accelerates warehouse analysis while it reduces database server CPU consumption and the I/O load between the database server and storage servers.



Smart Scan Requirements

Smart Scan is not governed by the optimizer, but it is influenced by the results of query optimization.

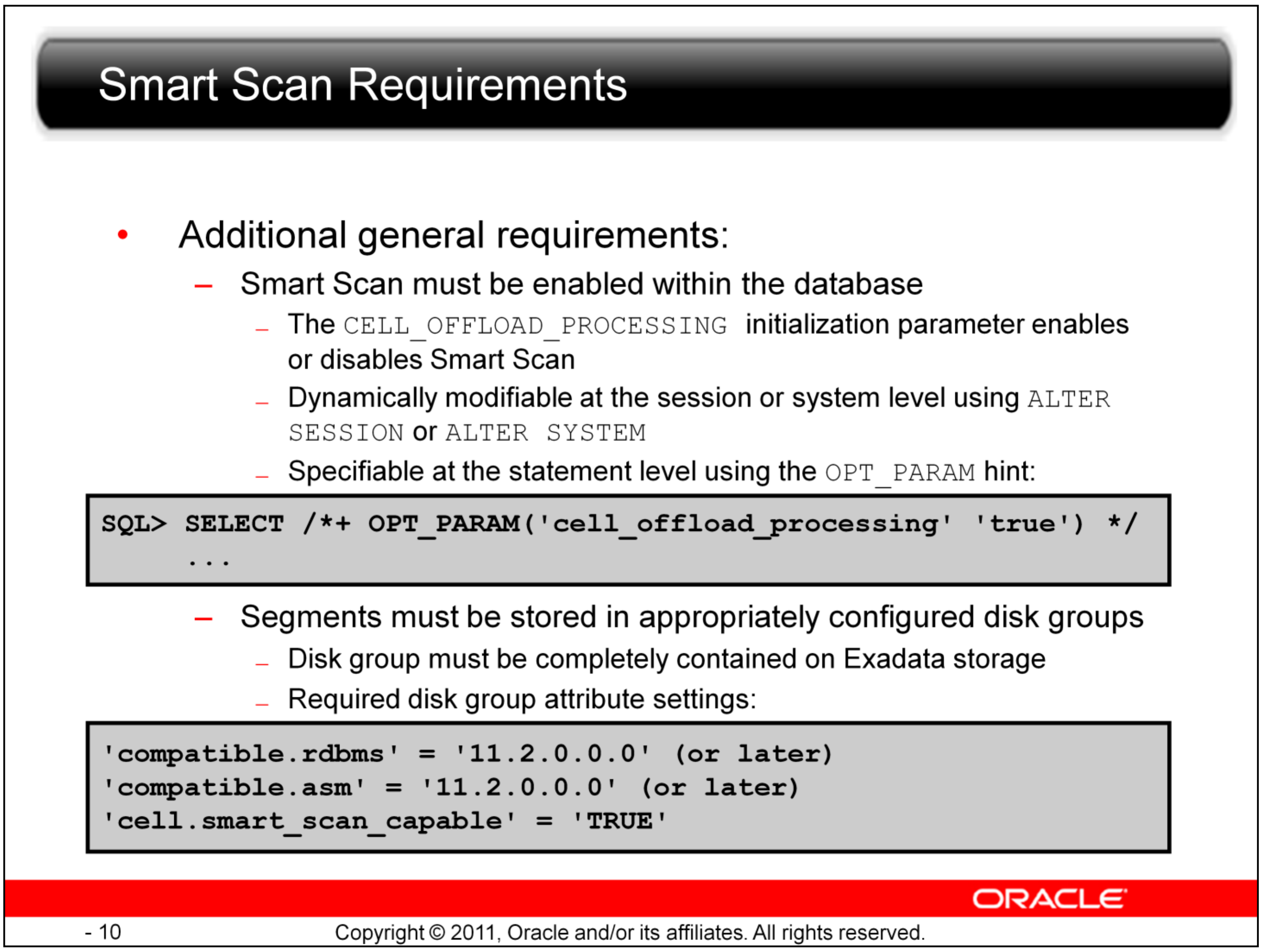
- Query-specific requirements:
  - Smart Scan is possible only for full table or index scans
  - Smart Scan can only be used for direct-path reads
    - Direct-path reads are automatically used for parallel queries
    - Direct-path reads may be used for serial queries
      - Not used by default for serial small table scans
      - Use `_serial_direct_read=TRUE` to force direct-path reads

- 9 Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Smart Scan optimization is a run-time decision. It is not integrated with the Oracle optimizer; however, it is influenced by the results of query optimization. In order words, the decision regarding whether or not to use Smart Scan is not made by the optimizer, but the optimizer does indirectly determine when Smart Scan can be used.

The following query-specific requirements must be met before Smart Scan is considered:

- Smart Scan is possible only for full segment scans; that is, full table scans, fast full index scans and fast full bitmap index scans.
- Smart Scan can only be used in conjunction with direct-path reads. Direct path reads are used by parallel operations so any parallel query is automatically a potential candidate for Smart Scan. Serial operations can do direct reads too, depending on factors such as the table size and the state of the buffer cache. By default, direct-path reads are not used for tables that are considered to be small. However, direct-path reads can be forced for serial access by setting `_serial_direct_read=TRUE` at either the system or session level.



Smart Scan Requirements

- Additional general requirements:
  - Smart Scan must be enabled within the database
    - The `CELL_OFFLOAD_PROCESSING` initialization parameter enables or disables Smart Scan
    - Dynamically modifiable at the session or system level using `ALTER SESSION` or `ALTER SYSTEM`
    - Specifiable at the statement level using the `OPT_PARAM` hint:

```
SQL> SELECT /*+ OPT_PARAM('cell_offload_processing' 'true') */
     ...
```

  - Segments must be stored in appropriately configured disk groups
    - Disk group must be completely contained on Exadata storage
    - Required disk group attribute settings:

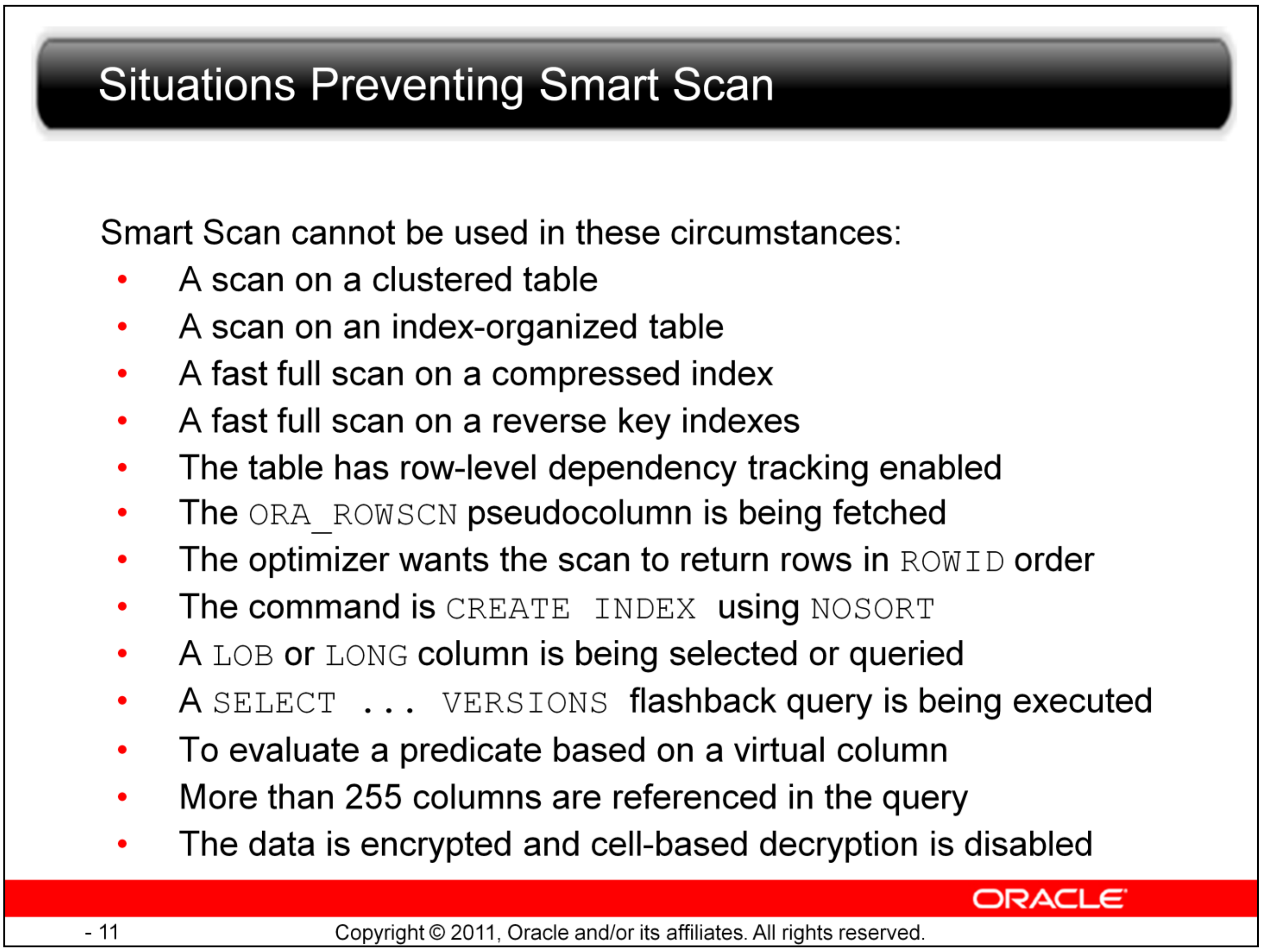
```
'compatible.rdbms' = '11.2.0.0.0' (or later)
'compatible.asm' = '11.2.0.0.0' (or later)
'cell.smart_scan_capable' = 'TRUE'
```

ORACLE

- 10

In addition to the query-specific requirements, the following general requirements must also be met to enable Smart Scan:

- Smart Scan must be enabled within the database. The `CELL_OFFLOAD_PROCESSING` initialization parameter controls Smart Scan. The default value of the parameter is `TRUE`, meaning that Smart Scan is enabled by default. If it is set to `FALSE`, Smart Scan is disabled and the database uses Exadata storage to serve data blocks similar to traditional storage. To enable Smart Scan for a particular SQL statement, use the `OPT_PARAM` hint as shown in the query fragment on the screen.
- Each segment being scanned must be on a disk group that is completely stored on Exadata cells. The disk group must also have the disk group attribute settings shown at the bottom of the screen.



Situations Preventing Smart Scan

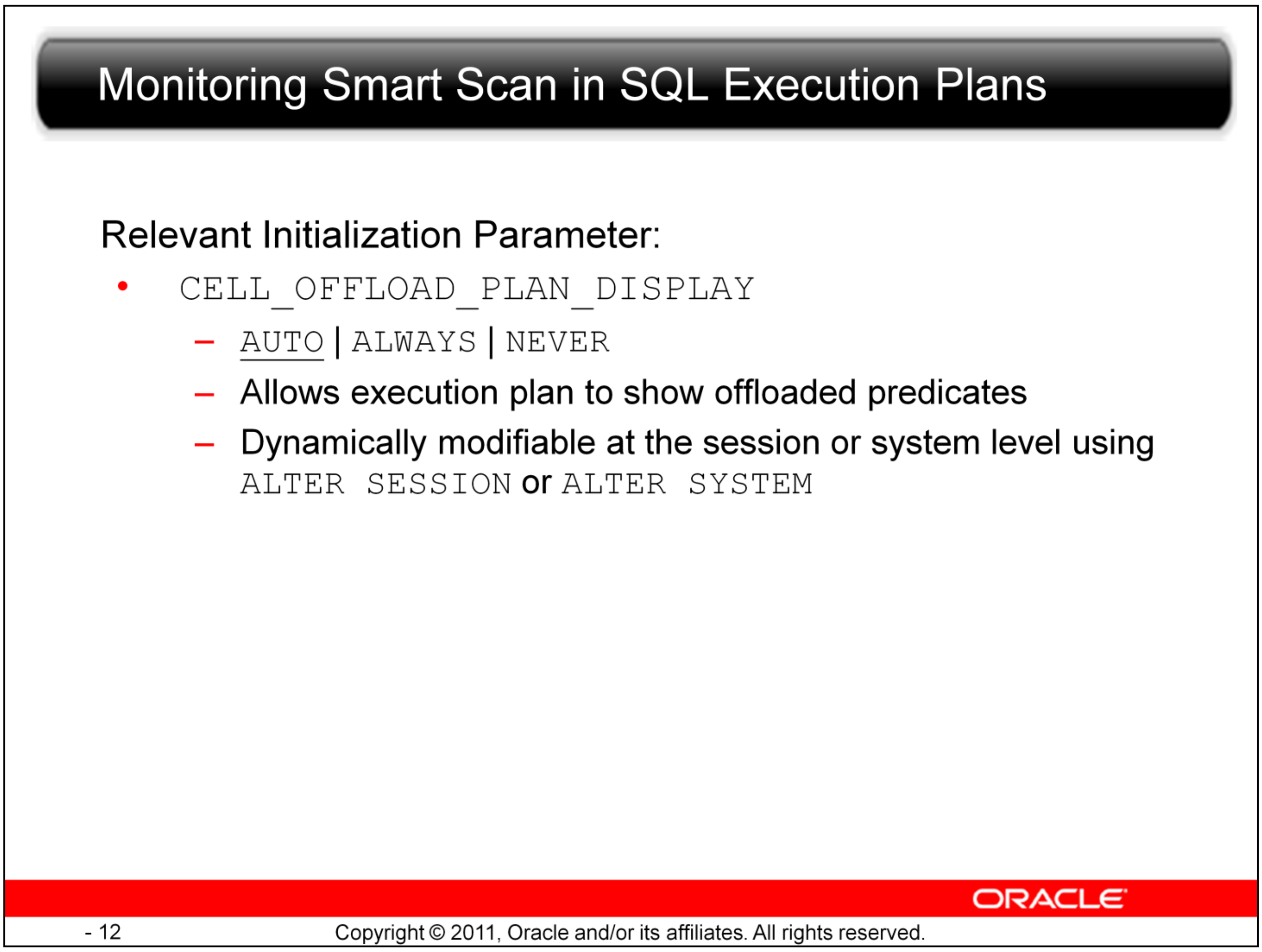
Smart Scan cannot be used in these circumstances:

- A scan on a clustered table
- A scan on an index-organized table
- A fast full scan on a compressed index
- A fast full scan on a reverse key indexes
- The table has row-level dependency tracking enabled
- The `ORA_ROWSCN` pseudocolumn is being fetched
- The optimizer wants the scan to return rows in `ROWID` order
- The command is `CREATE INDEX` using `NOSORT`
- A `LOB` or `LONG` column is being selected or queried
- A `SELECT ... VERSIONS` flashback query is being executed
- To evaluate a predicate based on a virtual column
- More than 255 columns are referenced in the query
- The data is encrypted and cell-based decryption is disabled

- 11

This slide lists specific circumstances where Smart Scan cannot be used. Most of these are pretty straight-forward; however, the final two scenarios on the list are worthy of further explanation:

- More than 255 columns are referenced in the query: This restriction only applies if the query involves tables that are not compressed using Exadata Hybrid Columnar Compression. Queries on tables compressed using Exadata Hybrid Columnar Compression can be offloaded even if they reference more than 255 columns.
- The data is encrypted and cell-based decryption is disabled: In order for Exadata Storage Server to perform decryption, Oracle Database needs to send the decryption key to each cell. If there are security concerns about keys being shipped across the storage network, cell-based decryption can be disabled by setting the `CELL_OFFLOAD_DECRYPTION` initialization parameter to `FALSE`.



Monitoring Smart Scan in SQL Execution Plans

Relevant Initialization Parameter:

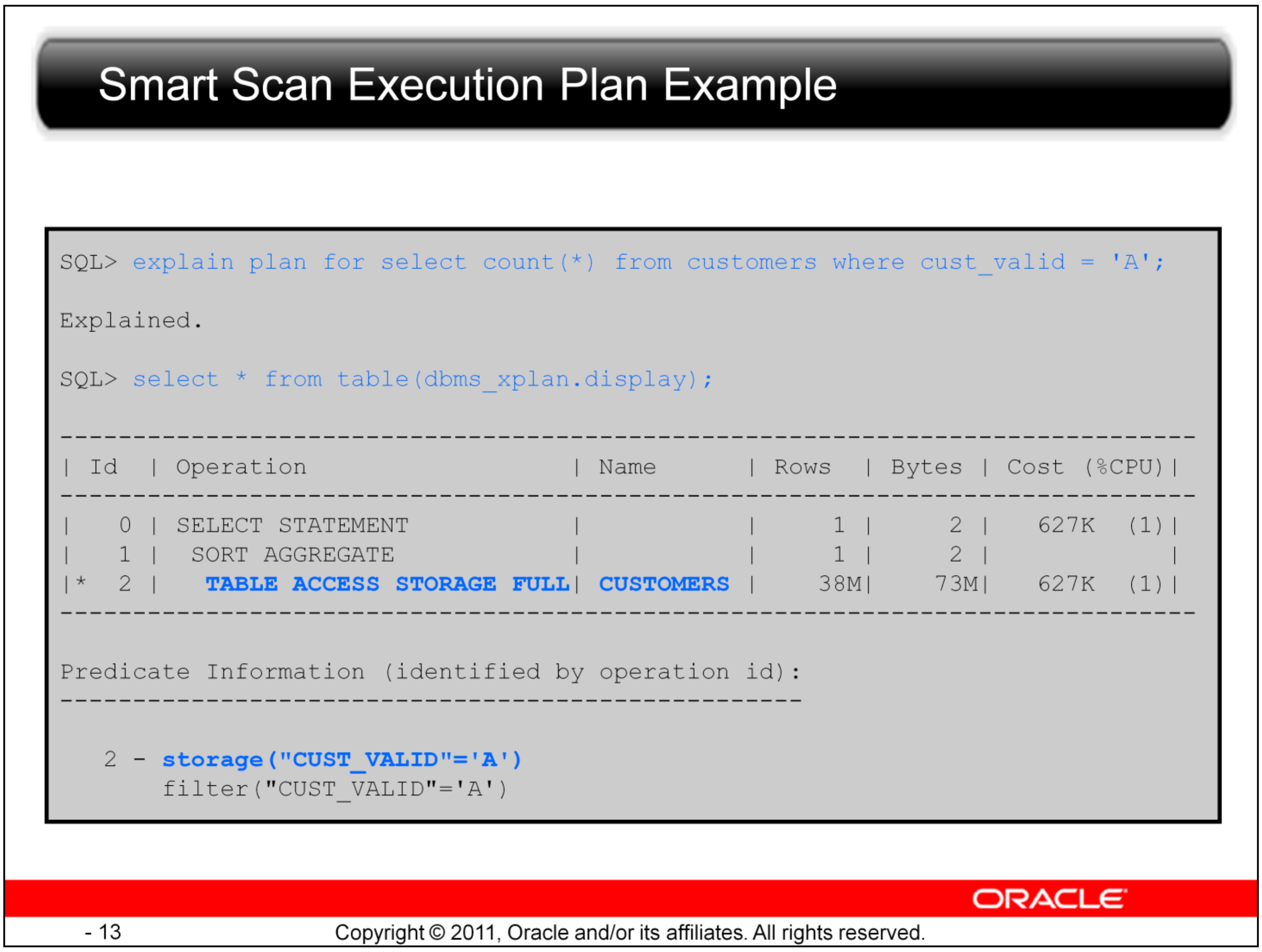
- CELL_OFFLOAD_PLAN_DISPLAY
  - AUTO | ALWAYS | NEVER
  - Allows execution plan to show offloaded predicates
  - Dynamically modifiable at the session or system level using ALTER SESSION or ALTER SYSTEM

ORACLE

- 12 Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The CELL_OFFLOAD_PLAN_DISPLAY initialization parameter determines whether the SQL EXPLAIN PLAN statement displays the predicates that can be evaluated by Exadata Storage Server as STORAGE predicates for a given SQL statement. The possible values are:

- AUTO instructs the SQL EXPLAIN PLAN statement to display the predicates that can be evaluated as STORAGE only if a cell is present and if a table is on the cell. AUTO is the default setting.
- ALWAYS produces changes to the SQL EXPLAIN PLAN statement output whether or not Exadata storage is present or the table is on the cell. You can use this setting to identify statements that are candidates for offloading before migrating to Exadata Database Machine.
- NEVER produces no changes to the SQL EXPLAIN PLAN statement output due to Exadata. This may be desirable, for example, if you wrote tools that process execution plan output and these tools have not been updated to deal with the updated syntax, or when comparing plans from Database Machine with plans from your previous system.



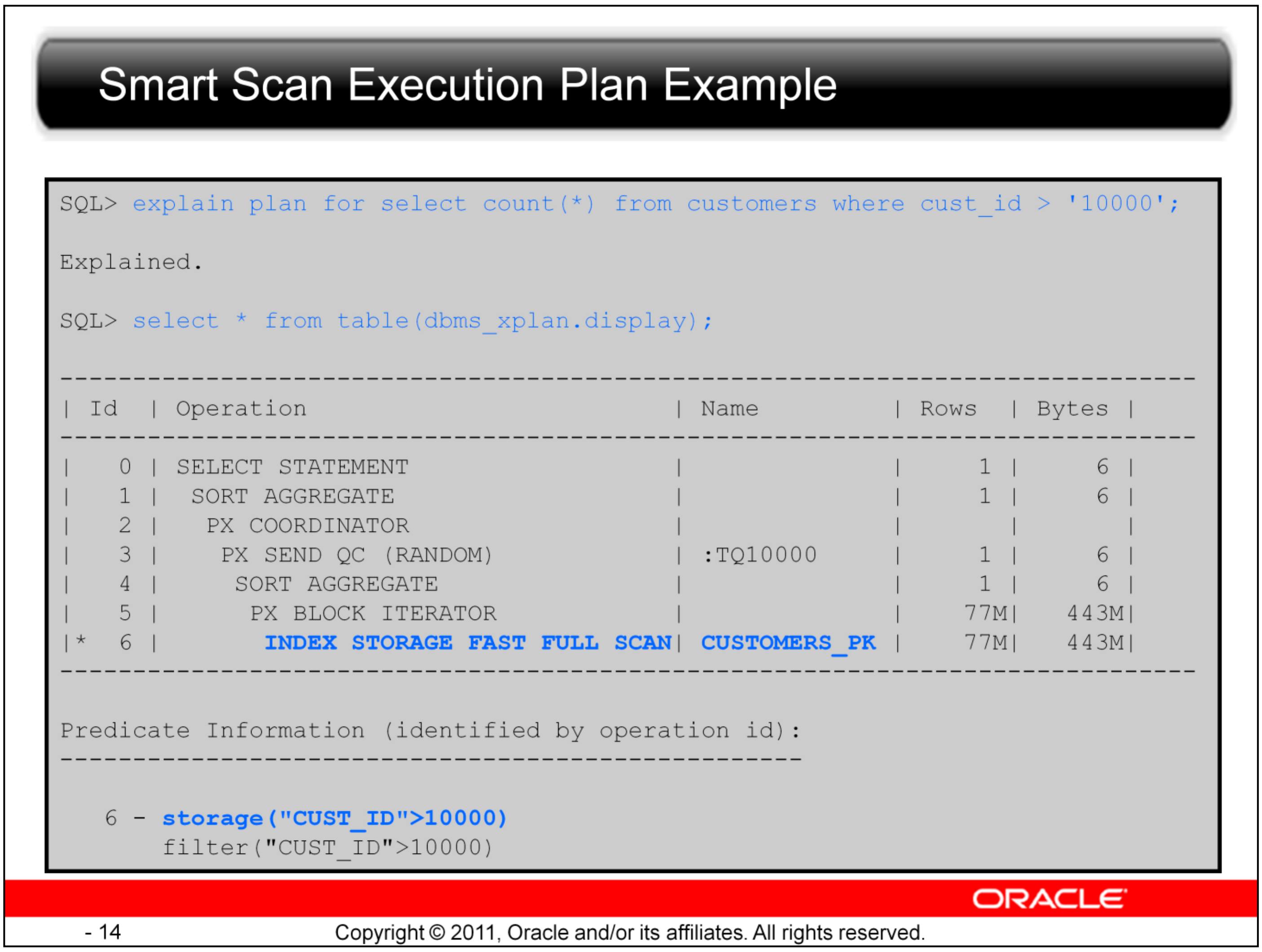
Smart Scan Execution Plan Example

```
SQL> explain plan for select count(*) from customers where cust_valid = 'A';

Explained.

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

-------------------------------------------------------------------------------
| Id  | Operation                  | Name      | Rows  | Bytes | Cost  (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |           |     1 |     2 |   627K  (1)|
|   1 |  SORT AGGREGATE            |           |     1 |     2 |            |
|*  2 |   TABLE ACCESS STORAGE FULL| CUSTOMERS |    38M|    73M|   627K  (1)|
-------------------------------------------------------------------------------

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

   2 - storage("CUST_VALID"='A')
       filter("CUST_VALID"='A')
```

The slide shows a basic example of Smart Scan manifested in a query plan.

The `TABLE ACCESS STORAGE FULL` operation indicates Smart Scan being used to scan the CUSTOMERS table. The plan also shows evidence of row filtering. In this example, the predicate `"CUST_VALID"='A'` is evaluated inside Exadata Storage Server.

Note that the execution plan shows no evidence of parallel query, so evidently the table in this example is large enough to prompt the use of direct-path reads.



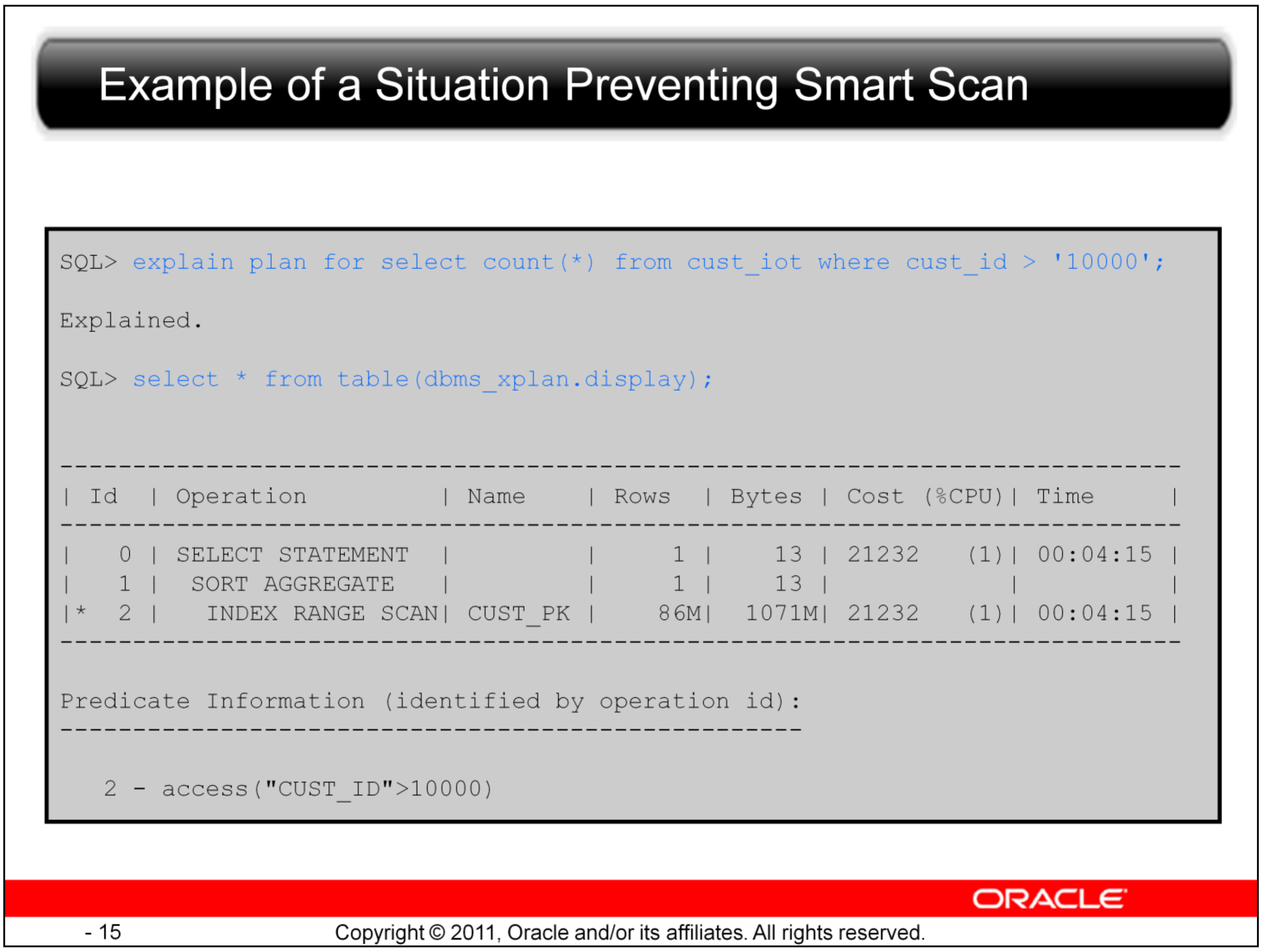
Smart Scan Execution Plan Example

```
SQL> explain plan for select count(*) from customers where cust_id > '10000';

Explained.

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

-----------------------------------------------------------------------------
| Id  | Operation                        | Name          | Rows   | Bytes |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT                 |               |      1 |      6 |
|   1 |  SORT AGGREGATE                  |               |      1 |      6 |
|   2 |   PX COORDINATOR                 |               |        |        |
|   3 |    PX SEND QC (RANDOM)           | :TQ10000      |      1 |      6 |
|   4 |     SORT AGGREGATE               |               |      1 |      6 |
|   5 |      PX BLOCK ITERATOR           |               |     77M|    443M|
|*  6 |       INDEX STORAGE FAST FULL SCAN| CUSTOMERS_PK |     77M|    443M|
-----------------------------------------------------------------------------

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

   6 - storage("CUST_ID">10000)
       filter("CUST_ID">10000)
```

ORACLE

This slide shows another Smart Scan query plan example. This time, offloading of a fast full index scan is indicated. Row filtering is also pushed down to the storage servers.

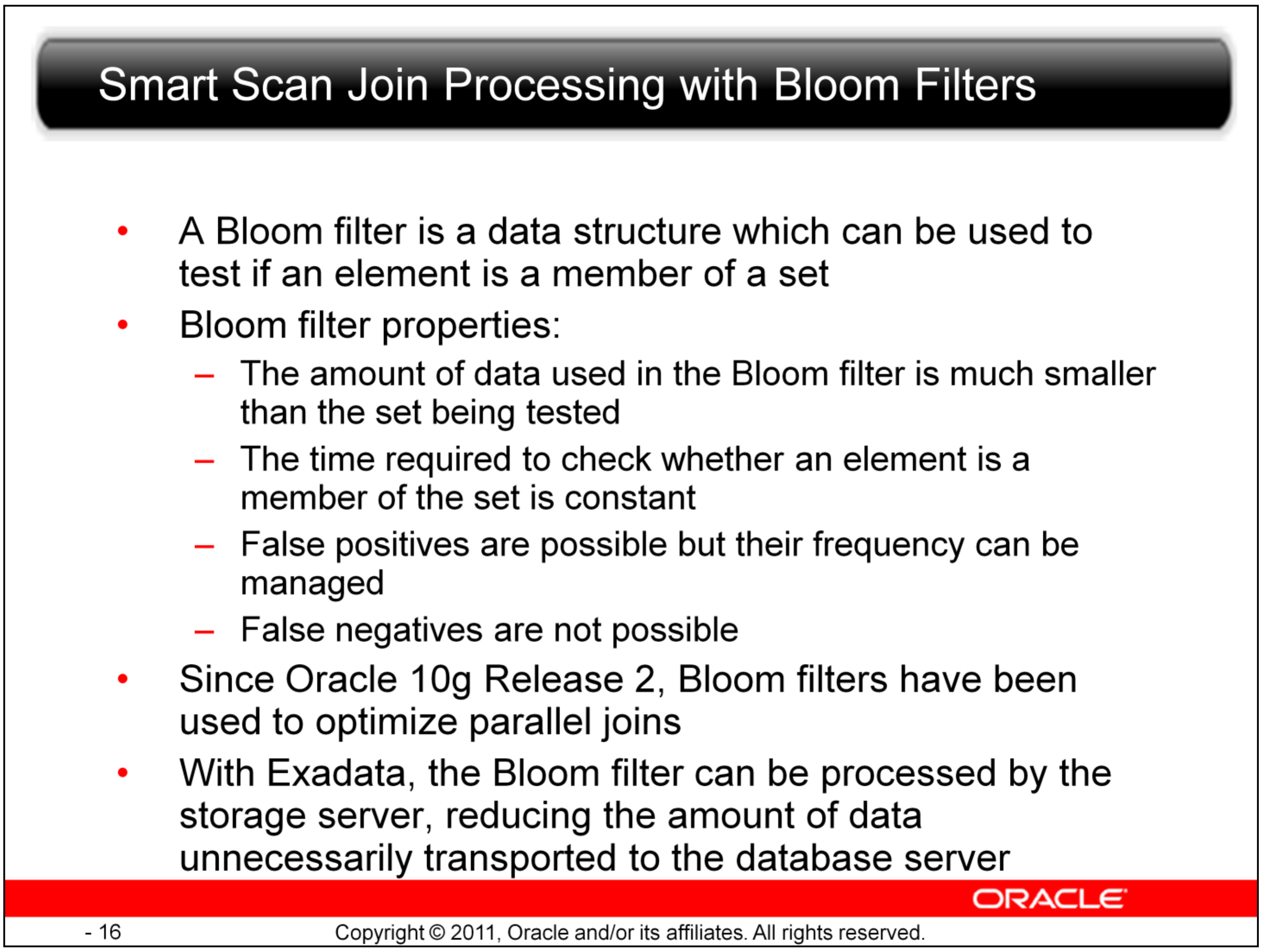
Unlike the previous example, this example indicates the use of parallel query which automatically implies the use of direct-path reads.



Example of a Situation Preventing Smart Scan

```
SQL> explain plan for select count(*) from cust_iot where cust_id > '10000';

Explained.

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


-----------------------------------------------------------------------------
| Id  | Operation           | Name    | Rows   | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |         |      1 |    13 | 21232   (1)| 00:04:15 |
|   1 |  SORT AGGREGATE     |         |      1 |    13 |            |          |
|*  2 |   INDEX RANGE SCAN| CUST_PK |    86M|  1071M| 21232   (1)| 00:04:15 |
-----------------------------------------------------------------------------

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

   2 - access("CUST_ID">10000)
```

This slide shows almost exactly the same example as the previous slide. The only difference is that the table being queried in this example is an index-organized table with the `CUST_ID` column defined as the primary key. Since Smart Scan cannot be applied to index-organized tables the execution plan contains no storage operations.



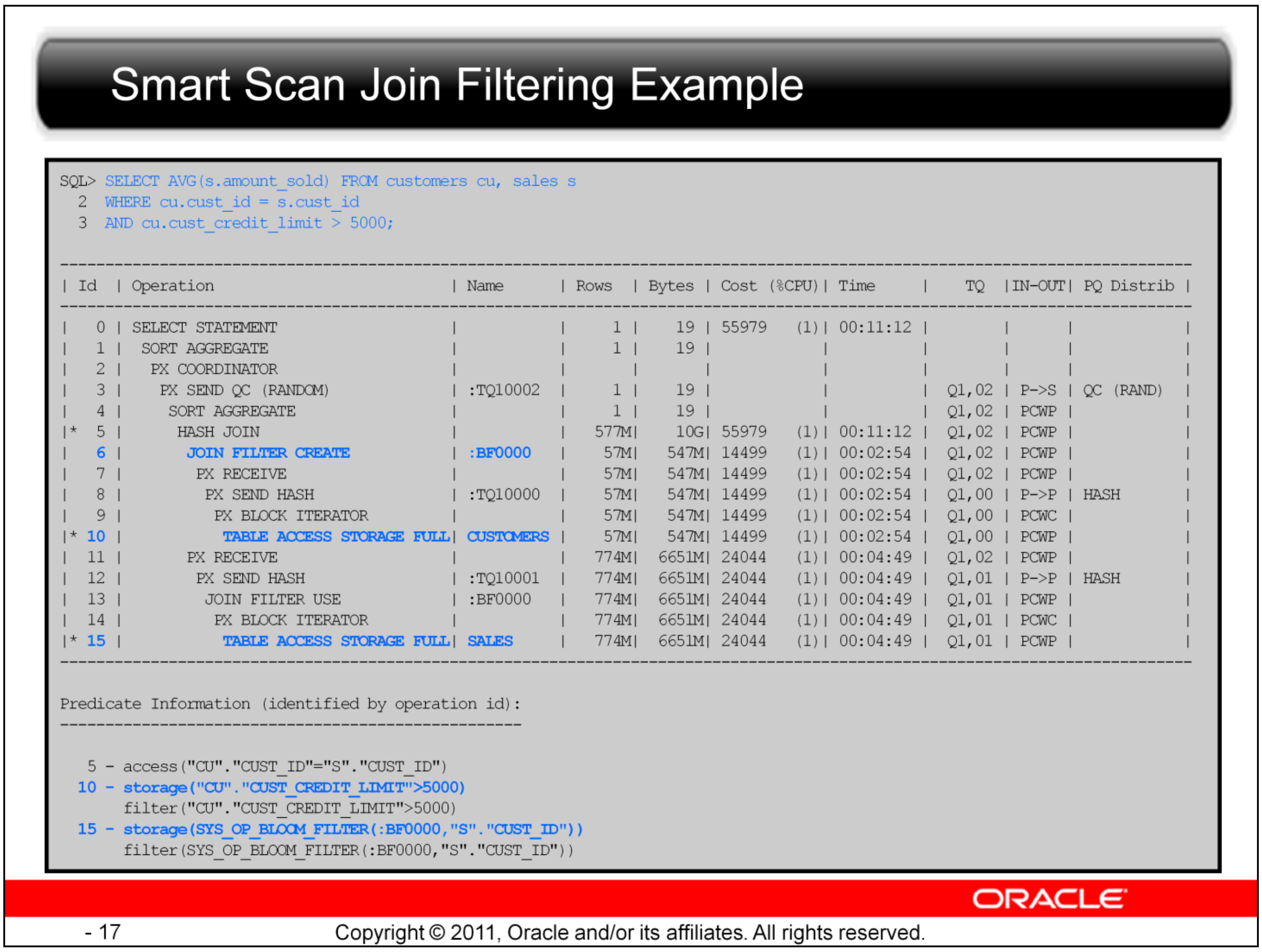
Smart Scan Join Processing with Bloom Filters

- A Bloom filter is a data structure which can be used to test if an element is a member of a set
- Bloom filter properties:
  - The amount of data used in the Bloom filter is much smaller than the set being tested
  - The time required to check whether an element is a member of the set is constant
  - False positives are possible but their frequency can be managed
  - False negatives are not possible
- Since Oracle 10g Release 2, Bloom filters have been used to optimize parallel joins
- With Exadata, the Bloom filter can be processed by the storage server, reducing the amount of data unnecessarily transported to the database server

A Bloom filter, conceived by Burton Howard Bloom in 1970, is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set. The properties of a Bloom filter make is a very efficient way of determining which values are not in a set. This is very useful for processing join conditions where a significant proportion of the data does not fulfill the join criteria.

Oracle Database 10g Release 2 first used Bloom filters to optimize parallel join operations. When two tables are joined via a hash join, the first table (typically the smaller table) is scanned and the rows that satisfy the `WHERE` clause predicates (for that table) are used to create a hash table. During the hash table creation, a Bloom filter bit string is also created based on the join column. The bit string is then sent as an additional predicate to the second table scan. After the `WHERE` clause predicates relating to the second table are applied, the resulting rows are tested using the Bloom filter. Any rows rejected by the Bloom filter must fail the join criteria and are discarded. Any rows that match using the Bloom filter are sent to the hash join.

With Exadata, the Bloom filter is passed to the storage servers as an additional predicate. Processing the Bloom filter inside Exadata Storage Server can reduce the amount of data transported to the database server to process a join, which in turn can speed up query performance.



Smart Scan Join Filtering Example

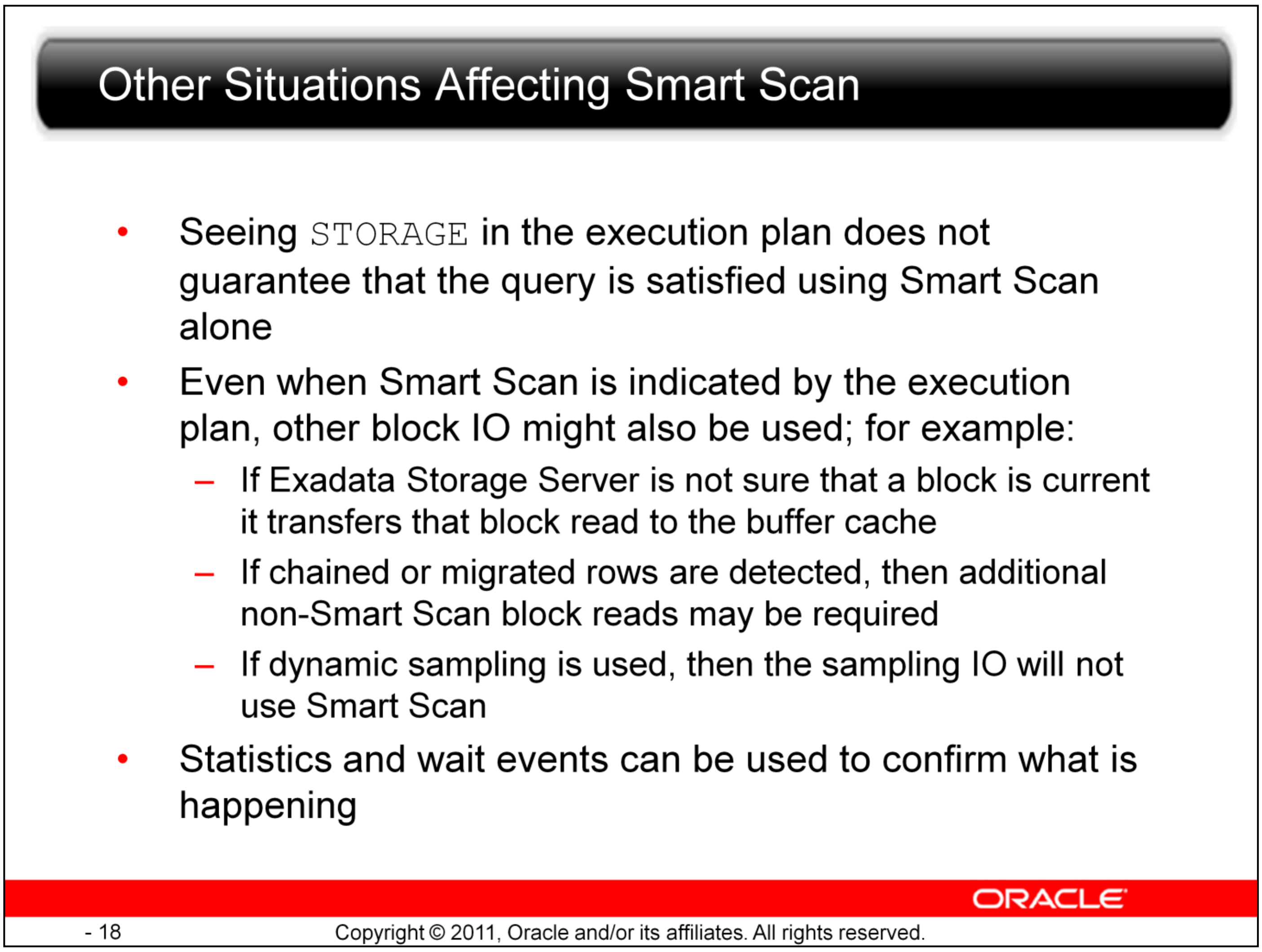
```
SQL> SELECT AVG(s.amount_sold) FROM customers cu, sales s
  2  WHERE cu.cust_id = s.cust_id
  3  AND cu.cust_credit_limit > 5000;

----------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name     | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
----------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |          |     1 |    19 | 55979   (1)| 00:11:12 |        |      |            |
|   1 |  SORT AGGREGATE                    |          |     1 |    19 |            |          |        |      |            |
|   2 |   PX COORDINATOR                   |          |       |       |            |          |        |      |            |
|   3 |    PX SEND QC (RANDOM)             | :TQ10002 |     1 |    19 |            |          |  Q1,02 | P->S | QC (RAND)  |
|   4 |     SORT AGGREGATE                 |          |     1 |    19 |            |          |  Q1,02 | PCWP |            |
|*  5 |      HASH JOIN                     |          |   577M|    10G| 55979   (1)| 00:11:12 |  Q1,02 | PCWP |            |
|   6 |       JOIN FILTER CREATE           | :BF0000  |    57M|   547M| 14499   (1)| 00:02:54 |  Q1,02 | PCWP |            |
|   7 |        PX RECEIVE                  |          |    57M|   547M| 14499   (1)| 00:02:54 |  Q1,02 | PCWP |            |
|   8 |         PX SEND HASH               | :TQ10000 |    57M|   547M| 14499   (1)| 00:02:54 |  Q1,00 | P->P | HASH       |
|   9 |          PX BLOCK ITERATOR         |          |    57M|   547M| 14499   (1)| 00:02:54 |  Q1,00 | PCWC |            |
|* 10 |           TABLE ACCESS STORAGE FULL| CUSTOMERS |   57M|   547M| 14499   (1)| 00:02:54 |  Q1,00 | PCWP |            |
|  11 |       PX RECEIVE                   |          |   774M|  6651M| 24044   (1)| 00:04:49 |  Q1,02 | PCWP |            |
|  12 |        PX SEND HASH                | :TQ10001 |   774M|  6651M| 24044   (1)| 00:04:49 |  Q1,01 | P->P | HASH       |
|  13 |         JOIN FILTER USE            | :BF0000  |   774M|  6651M| 24044   (1)| 00:04:49 |  Q1,01 | PCWP |            |
|  14 |          PX BLOCK ITERATOR         |          |   774M|  6651M| 24044   (1)| 00:04:49 |  Q1,01 | PCWC |            |
|* 15 |           TABLE ACCESS STORAGE FULL| SALES    |   774M|  6651M| 24044   (1)| 00:04:49 |  Q1,01 | PCWP |            |
----------------------------------------------------------------------------------------------------------------------------

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

   5 - access("CU"."CUST_ID"="S"."CUST_ID")
  10 - storage("CU"."CUST_CREDIT_LIMIT">5000)
       filter("CU"."CUST_CREDIT_LIMIT">5000)
  15 - storage(SYS_OP_BLOOM_FILTER(:BF0000,"S"."CUST_ID"))
       filter(SYS_OP_BLOOM_FILTER(:BF0000,"S"."CUST_ID"))
```

ORACLE

- 17

The slide shows an example of Smart Scan join filtering using a Bloom filter. The example query joins a table containing approximately 77 million customer records, with a table containing approximately 774 million sales records, to determine the average transaction amount for customers having a credit limit in excess of 5000. The query is processed as follows:

- Smart Scan is used to filter the customer records and retrieve the `CUST_ID` values for the customers having a credit limit in excess of 5000 (operation 10).
- A Bloom filter is created representing the set of `CUST_ID` values, which match the query from the `CUSTOMERS` table (operation 6).
- In the absence of Exadata storage, the Bloom filter would be applied to data from the `SALES` table inside a parallel query server process (operation 13). However, with Exadata, the Bloom filter is sent to the storage servers as a predicate and applied as part of a Smart Scan of the `SALES` table (operation 15). Since the `SALES` table is quite large, query performance will benefit considerably if a significant amount of IO is saved by not transporting unnecessary sales records back to the database. Exadata storage server also performs column filtering for the `SALES` table so that only the `CUST_ID` and `AMOUNT_SOLD` values are returned.
- The results from the Smart Scan operations on the `CUSTOMERS` table (operations 6-10) and the `SALES` table (operations 11-15) are combined using a `HASH JOIN` (operation 5).
- Finally, the query is serialized and the result is returned (operations 1-4).



Other Situations Affecting Smart Scan

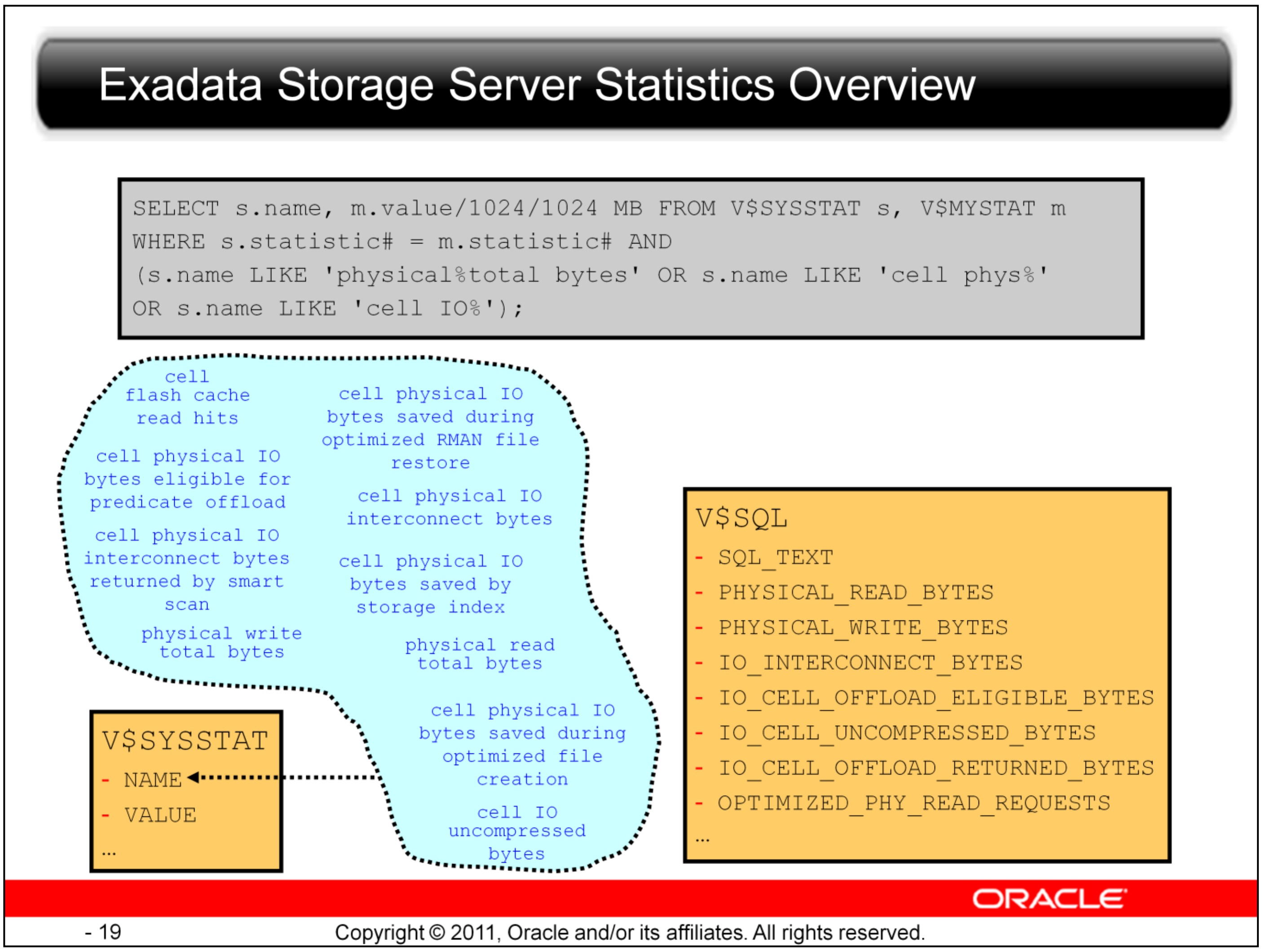
- Seeing STORAGE in the execution plan does not guarantee that the query is satisfied using Smart Scan alone
- Even when Smart Scan is indicated by the execution plan, other block IO might also be used; for example:
  - If Exadata Storage Server is not sure that a block is current it transfers that block read to the buffer cache
  - If chained or migrated rows are detected, then additional non-Smart Scan block reads may be required
  - If dynamic sampling is used, then the sampling IO will not use Smart Scan
- Statistics and wait events can be used to confirm what is happening

- 18

As we have already seen, the query execution plan provides an indicator regarding the use of Smart Scan; however, seeing a STORAGE operation in the plan does not guarantee that the query is satisfied entirely using Smart Scan. Beware that even when Smart Scan is indicated in the execution plan, other block IO might also be used. Here are some situations where this might occur:

- Since Smart Scan uses direct-path reads, the data being read must be current. If Exadata Storage Server is not sure that a block is current, it transfers the read of that block to the traditional buffer cache read-consistency path. So, if you run updates at the same time as queries you will benefit less from Smart Scan.
- The same is also true for indirect rows; that is, reads on chained or migrated rows. To resolve indirect row references, additional block reads may be required. So if your tables contain chained or migrated rows you will benefit less from Smart Scan.
- If the optimizer uses dynamic sampling to formulate the query execution plan, then the sampling IO does not use Smart Scan even if the query does end up using Smart Scan. In this case, you may see Smart Scan mixed in with other block IO.

Statistics and wait events can be used to confirm and measure the use of Smart Scan. A series of examples are presented later in the course.



Exadata Storage Server Statistics Overview

```
SELECT s.name, m.value/1024/1024 MB FROM V$SYSSTAT s, V$MYSTAT m
WHERE s.statistic# = m.statistic# AND
(s.name LIKE 'physical%total bytes' OR s.name LIKE 'cell phys%'
OR s.name LIKE 'cell IO%');
```

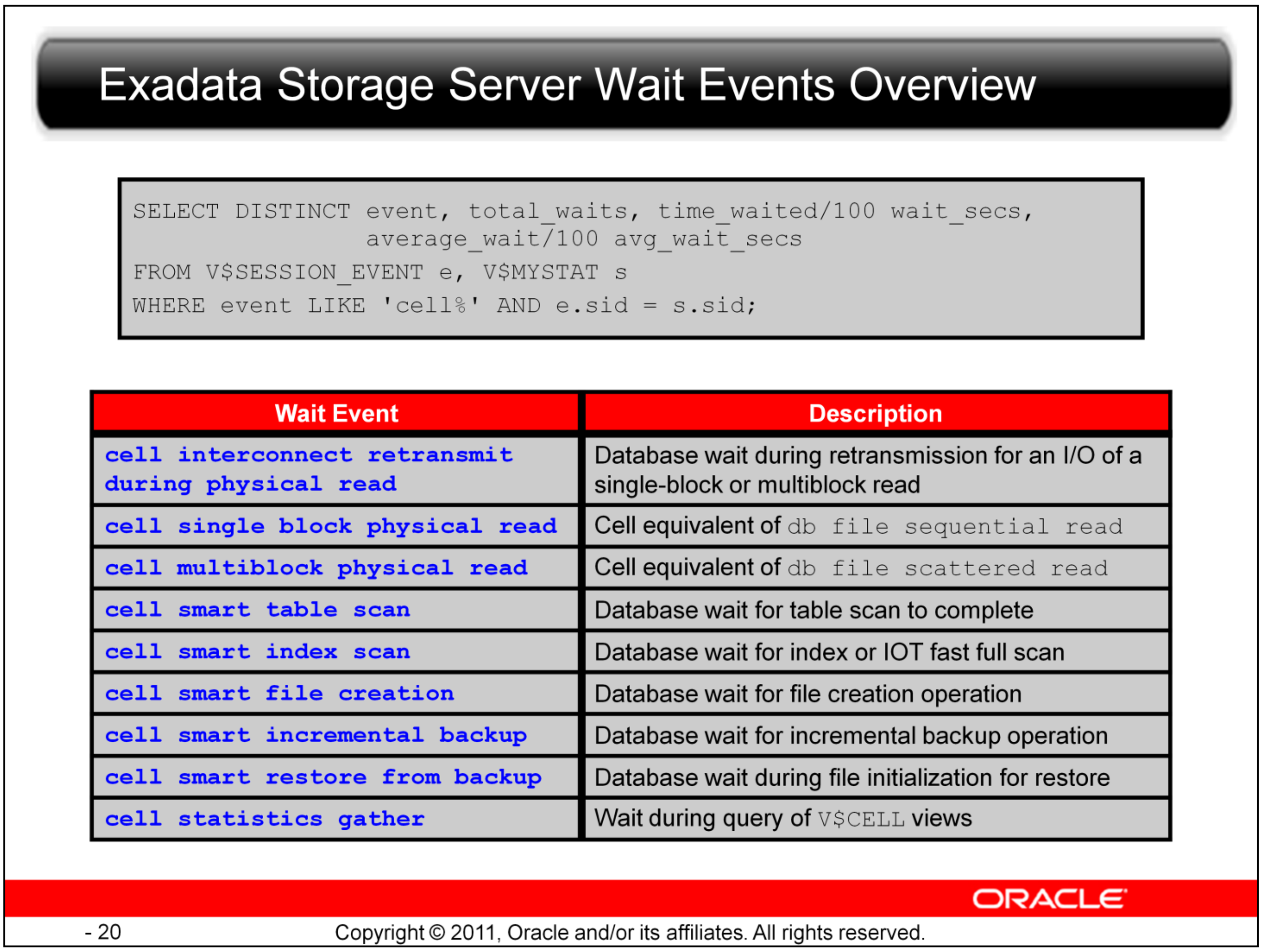
cell flash cache read hits
cell physical IO bytes eligible for predicate offload
cell physical IO interconnect bytes returned by smart scan
physical write total bytes
cell physical IO bytes saved during optimized RMAN file restore
cell physical IO interconnect bytes
cell physical IO bytes saved by storage index
physical read total bytes
cell physical IO bytes saved during optimized file creation
cell IO uncompressed bytes
V$SYSSTAT
- NAME
- VALUE
...
V$SQL
- SQL_TEXT
- PHYSICAL_READ_BYTES
- PHYSICAL_WRITE_BYTES
- IO_INTERCONNECT_BYTES
- IO_CELL_OFFLOAD_ELIGIBLE_BYTES
- IO_CELL_UNCOMPRESSED_BYTES
- IO_CELL_OFFLOAD_RETURNED_BYTES
- OPTIMIZED_PHY_READ_REQUESTS
...
ORACLE

Numerous cell-specific statistics are recorded in `V$SYSSTAT` and `V$SESSTAT`. These statistics can be used to monitor Exadata Storage Server operations at both the system and session level. The statistics can be used to monitor the effectiveness of Smart Scan. There are also statistics relating to Exadata Smart Flash Cache, Exadata Hybrid Columnar Compression, storage index, fast file creation, and optimized incremental backups. In addition, other statistics provide the total volume of I/O exchanged over the interconnect and the total volume of physical disk reads and writes. The slide lists a selection of the available statistics.

The query in the slide shows key cell statistics for the current session. Examples of the output from this query are shown later in the course.

In addition, `V$SQL` lists statistics on shared SQL areas. It contains statement-level statistics for the volume of physical I/O (reads and writes), the volume of I/O exchanged over the interconnect, along with information relating to the effectiveness of Smart Scan and other Exadata Storage Server features.

For more information on cell-specific statistics, refer to the *Oracle Exadata Storage Server Software User's Guide*.



Exadata Storage Server Wait Events Overview

```
SELECT DISTINCT event, total_waits, time_waited/100 wait_secs,
                average_wait/100 avg_wait_secs
FROM V$SESSION_EVENT e, V$MYSTAT s
WHERE event LIKE 'cell%' AND e.sid = s.sid;
```

Wait Event	Description
`cell interconnect retransmit during physical read`	Database wait during retransmission for an I/O of a single-block or multiblock read
`cell single block physical read`	Cell equivalent of `db file sequential read`
`cell multiblock physical read`	Cell equivalent of `db file scattered read`
`cell smart table scan`	Database wait for table scan to complete
`cell smart index scan`	Database wait for index or IOT fast full scan
`cell smart file creation`	Database wait for file creation operation
`cell smart incremental backup`	Database wait for incremental backup operation
`cell smart restore from backup`	Database wait during file initialization for restore
`cell statistics gather`	Wait during query of `V$CELL` views

- 20

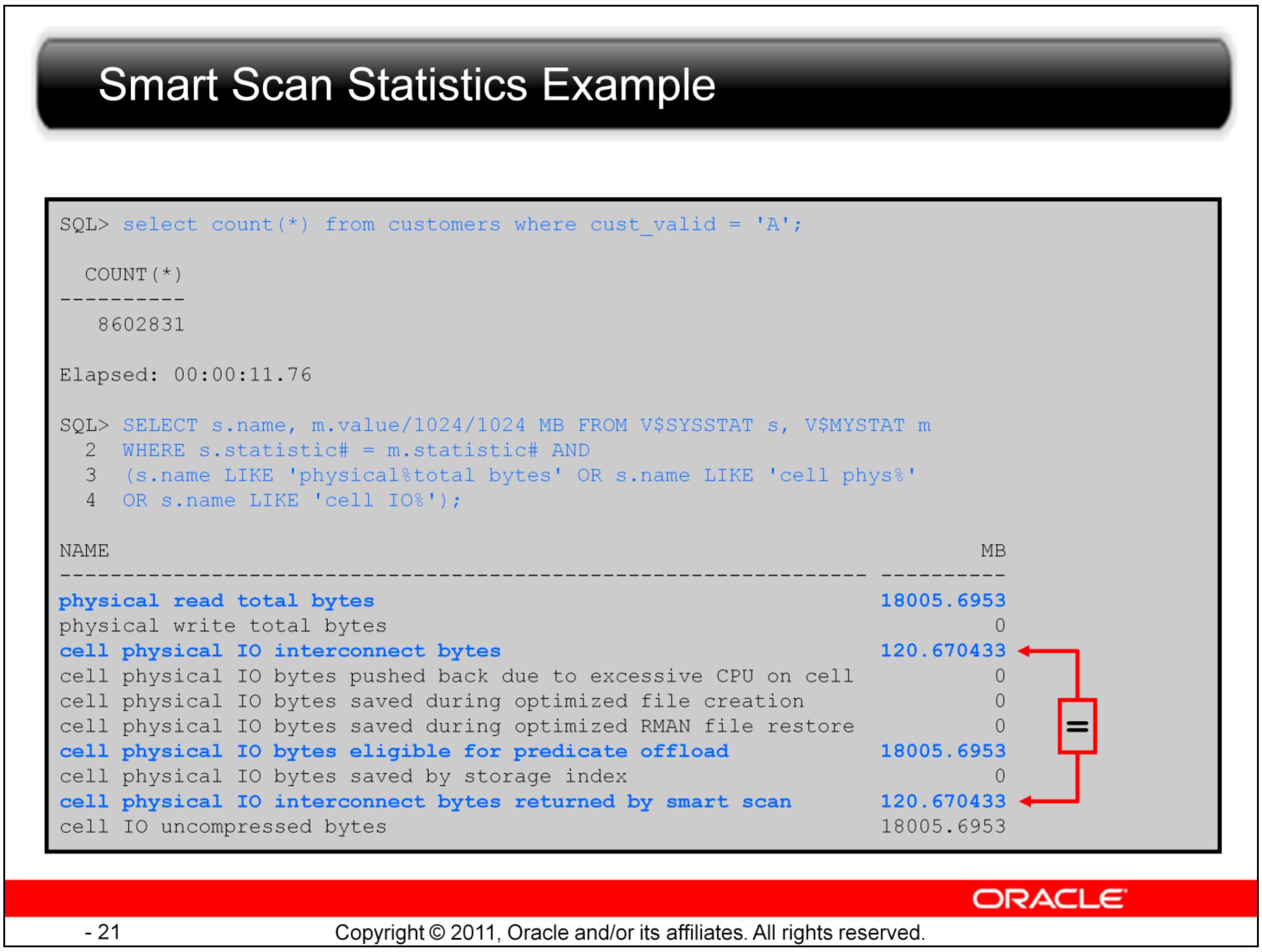
Oracle uses a specific set of wait events for disk I/O to Exadata Storage Server. Information about cell wait events is displayed in `V$` dynamic performance views, such as `V$SESSION_WAIT`, `V$SYSTEM_EVENT`, and `V$SESSION_EVENT`.

The slide shows an example of a query used to display a summary of cell wait events for the current session. Examples of the output from this query are shown later in the course.

A list of cell wait events with a brief description is also shown. Most of the cell wait events are self-explanatory however the `cell statistics gather` event is a little different. It appears when a query is done on the `V$CELL_STATE`, `V$CELL_THREAD_HISTORY`, or `V$CELL_REQUEST_TOTALS` view. During such a query, data from the cells and any wait events are shown in this wait event. Normally, these `V$CELL` views are only used by Oracle Support Services.

Note that for detailed analysis purposes, the cell wait events can also identify the corresponding cell and grid disk being accessed which is more useful for performance and diagnostics purposes than the database file number and block number information that is provided by wait events for conventional storage.

For more information about these wait events, refer to the *Oracle Exadata Storage Server Software User's Guide*.



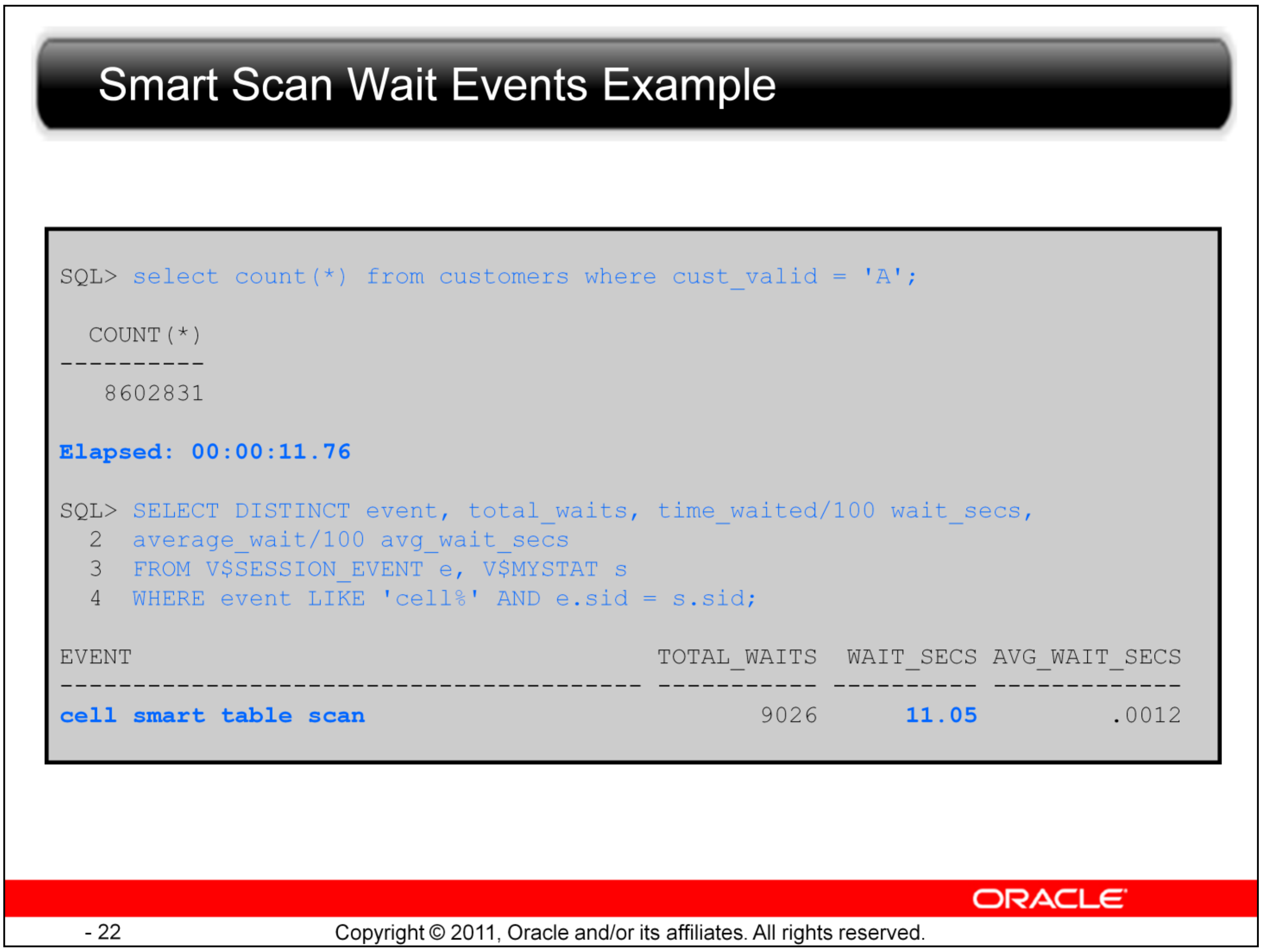
Smart Scan Statistics Example

```
SQL> select count(*) from customers where cust_valid = 'A';

  COUNT(*)
----------
   8602831

Elapsed: 00:00:11.76

SQL> SELECT s.name, m.value/1024/1024 MB FROM V$SYSSTAT s, V$MYSTAT m
  2  WHERE s.statistic# = m.statistic# AND
  3  (s.name LIKE 'physical%total bytes' OR s.name LIKE 'cell phys%'
  4  OR s.name LIKE 'cell IO%');

NAME                                                                    MB
---------------------------------------------------------------- ----------
physical read total bytes                                        18005.6953
physical write total bytes                                                0
cell physical IO interconnect bytes                              120.670433
cell physical IO bytes pushed back due to excessive CPU on cell           0
cell physical IO bytes saved during optimized file creation               0
cell physical IO bytes saved during optimized RMAN file restore           0
cell physical IO bytes eligible for predicate offload            18005.6953
cell physical IO bytes saved by storage index                             0
cell physical IO interconnect bytes returned by smart scan       120.670433
cell IO uncompressed bytes                                       18005.6953
```

ORACLE

This slide shows the first example query introduced earlier in the course. You might remember that the execution plan for this query indicated the use of Smart Scan. This is backed up by the statistics that follow the query. The statistics show that although the query referenced over 18,000 MB of data, Smart Scan reduced the amount of IO transported over the storage interconnect to a little more than 120 MB. In other words, Smart Scan reduced the amount of IO transported to the database server by more than 99% compared with what would have been required using non-Exadata storage.

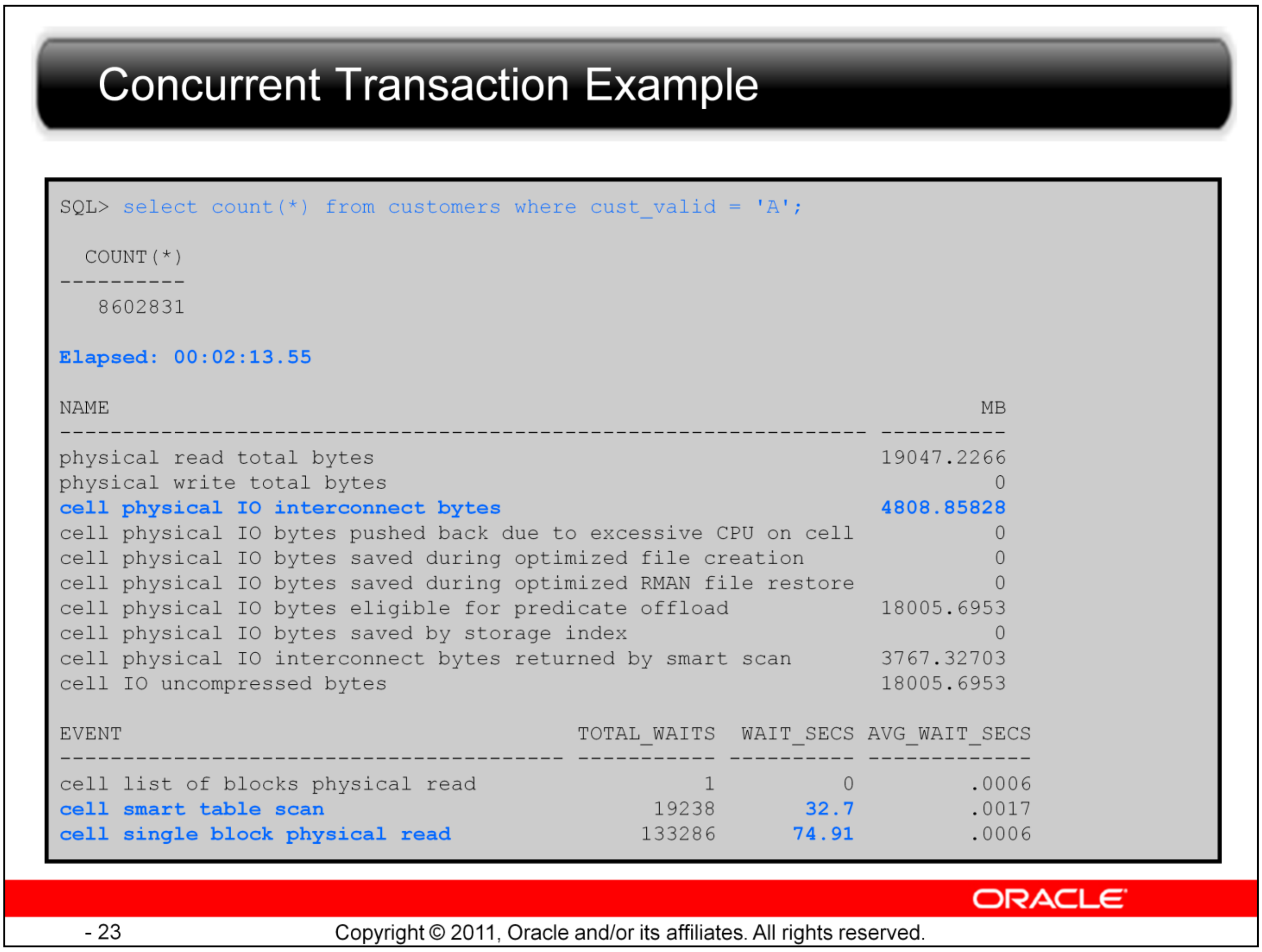
Note also that the amount of data returned by Smart Scan matches the amount of data transported across the storage interconnect. This is a sign that all the IO in this example is associated with Smart Scan.



Smart Scan Wait Events Example

```
SQL> select count(*) from customers where cust_valid = 'A';

  COUNT(*)
----------
   8602831

Elapsed: 00:00:11.76

SQL> SELECT DISTINCT event, total_waits, time_waited/100 wait_secs,
  2  average_wait/100 avg_wait_secs
  3  FROM V$SESSION_EVENT e, V$MYSTAT s
  4  WHERE event LIKE 'cell%' AND e.sid = s.sid;

EVENT                                    TOTAL_WAITS  WAIT_SECS AVG_WAIT_SECS
---------------------------------------- ----------- ---------- -------------
cell smart table scan                           9026      11.05         .0012
```

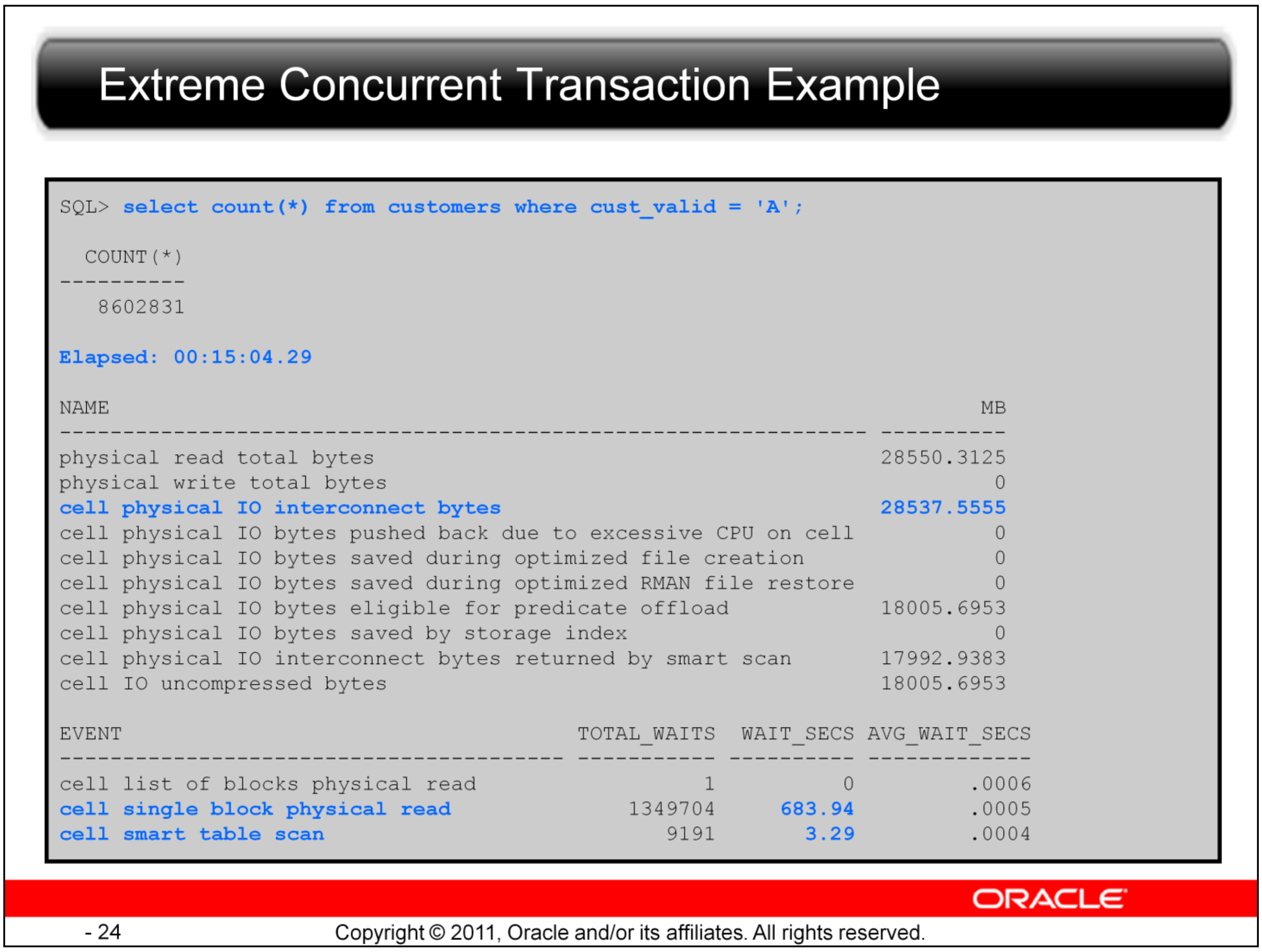
This slide shows the same example query are the previous slide, but this time the cell wait events associated with the query session are shown. The wait events confirm that all of the IO associated with the example query was satisfied using Smart Scan. Notice also that the Smart Scan total wait time (11.05 sec) accounts for almost all of the query elapsed time (11.76 sec). This means that nearly all of the processing for this query occurred inside the Exadata cells.



Concurrent Transaction Example

```
SQL> select count(*) from customers where cust_valid = 'A';

  COUNT(*)
----------
   8602831

Elapsed: 00:02:13.55

NAME                                                                    MB
---------------------------------------------------------------- ----------
physical read total bytes                                        19047.2266
physical write total bytes                                                0
cell physical IO interconnect bytes                              4808.85828
cell physical IO bytes pushed back due to excessive CPU on cell           0
cell physical IO bytes saved during optimized file creation               0
cell physical IO bytes saved during optimized RMAN file restore           0
cell physical IO bytes eligible for predicate offload            18005.6953
cell physical IO bytes saved by storage index                             0
cell physical IO interconnect bytes returned by smart scan       3767.32703
cell IO uncompressed bytes                                       18005.6953

EVENT                                    TOTAL_WAITS  WAIT_SECS AVG_WAIT_SECS
---------------------------------------- ----------- ---------- -------------
cell list of blocks physical read                  1          0         .0006
cell smart table scan                          19238       32.7         .0017
cell single block physical read               133286      74.91         .0006
```

The slide shows exactly the same query as before; however, this time a batch process was updating the CUSTOMERS table at the same time as the query. The wait events confirm that Smart Scan is still used; however, this time a large number of cell single block physical reads are also required. The statistics quantify the effect. Notice how the physical IO over the interconnect rises from approximately 120 MB in the previous example, to over 4800 MB in this case. Note also the increase in the query elapsed time and how it correlates with the wait times.



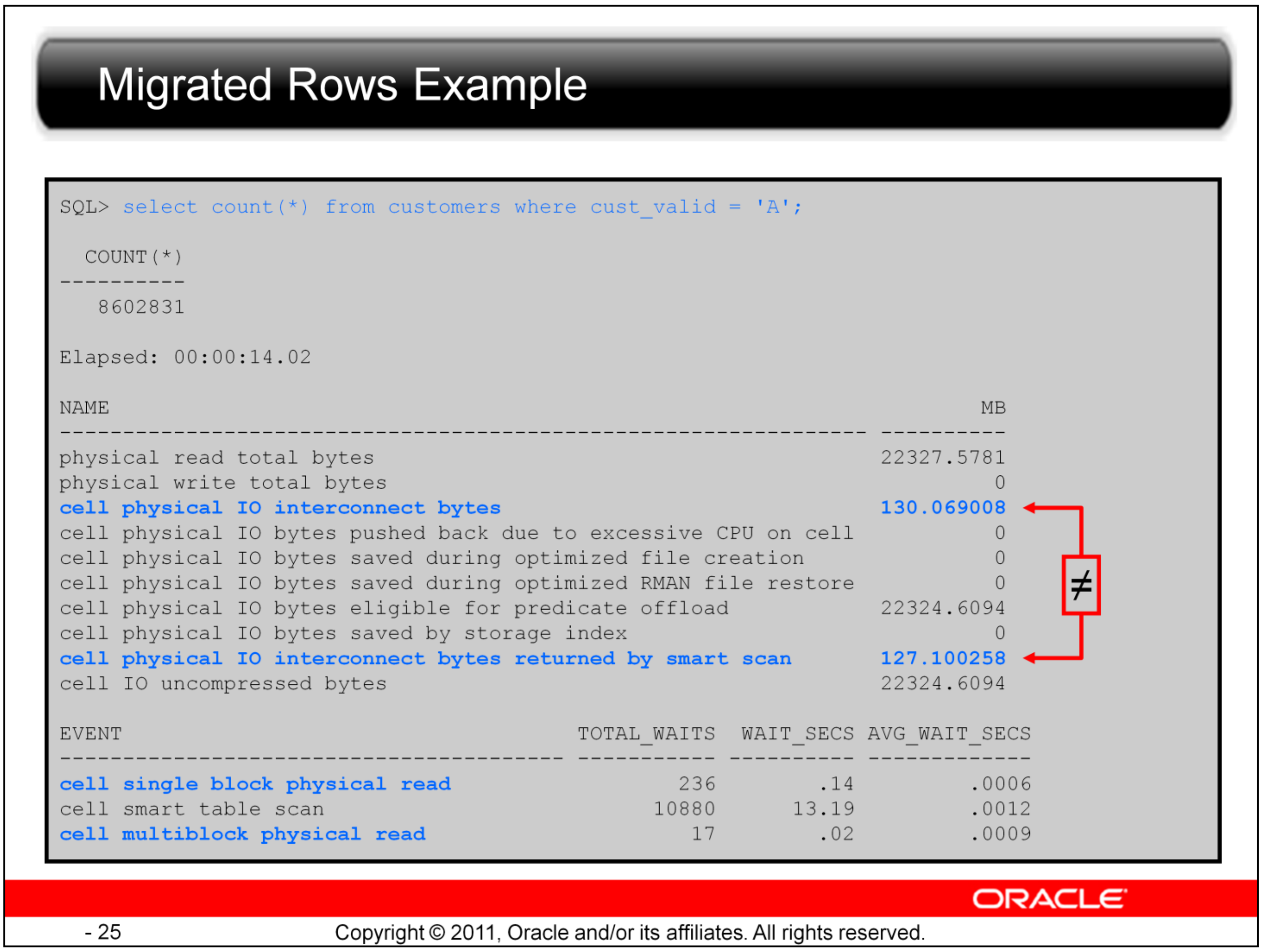
Extreme Concurrent Transaction Example

```
SQL> select count(*) from customers where cust_valid = 'A';

  COUNT(*)
----------
   8602831

Elapsed: 00:15:04.29

NAME                                                                    MB
---------------------------------------------------------------- ----------
physical read total bytes                                       28550.3125
physical write total bytes                                               0
cell physical IO interconnect bytes                             28537.5555
cell physical IO bytes pushed back due to excessive CPU on cell          0
cell physical IO bytes saved during optimized file creation              0
cell physical IO bytes saved during optimized RMAN file restore          0
cell physical IO bytes eligible for predicate offload           18005.6953
cell physical IO bytes saved by storage index                            0
cell physical IO interconnect bytes returned by smart scan      17992.9383
cell IO uncompressed bytes                                      18005.6953

EVENT                                    TOTAL_WAITS  WAIT_SECS AVG_WAIT_SECS
---------------------------------------- ----------- ---------- -------------
cell list of blocks physical read                  1          0         .0006
cell single block physical read              1349704     683.94         .0005
cell smart table scan                           9191       3.29         .0004
```

This time the query is executed after another session has updated every row in the CUSTOMERS table, but before the update transaction is committed (or rolled-back). In this extreme case Smart Scan is still attempted; however, since every record is subject to a pending transaction, every block I/O must be transferred to the traditional buffer cache read-consistency path. In this unusual case, attempting to use Smart Scan actually results in more I/O traffic across the storage interconnect than if Smart Scan was not used.

Following is a summary of the results for the same scenario, but with Smart Scan disabled:

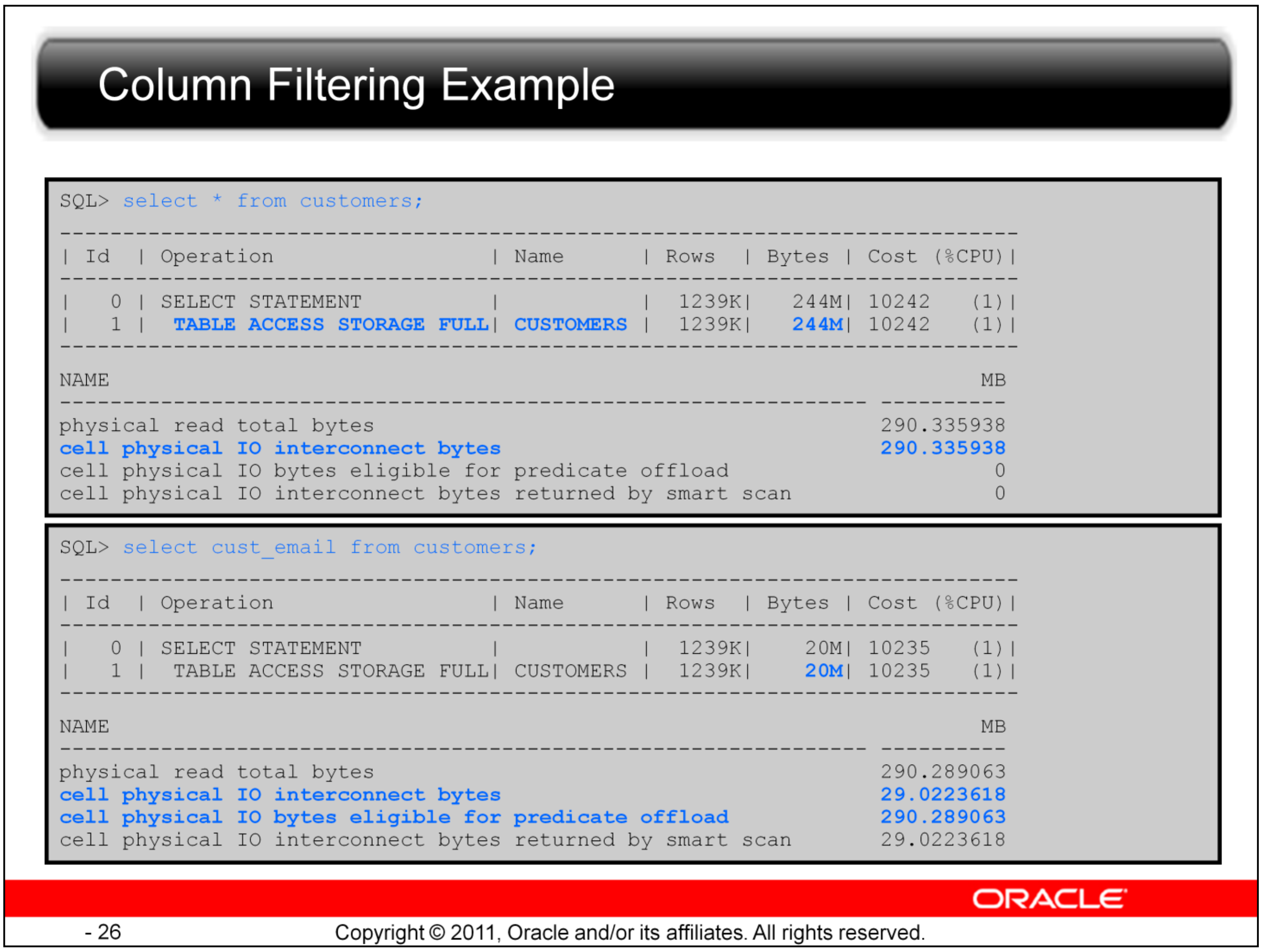
```
SQL> select /*+ OPT_PARAM('cell_offload_processing' 'false') */
count(*) from customers where cust_valid = 'A';

Elapsed: 00:14:52.63

NAME                                                           MB
------------------------------------------------------ ----------
cell physical IO interconnect bytes                    28522.4922
cell physical IO bytes eligible for predicate offload           0

EVENT                              TOTAL_WAITS WAIT_SECS AVG_WAIT
---------------------------------- ----------- --------- --------
cell single block physical read        1346130    678.83    .0005
cell list of blocks physical read            2         0    .0007
```



Migrated Rows Example

```
SQL> select count(*) from customers where cust_valid = 'A';

  COUNT(*)
----------
   8602831

Elapsed: 00:00:14.02

NAME                                                                     MB
---------------------------------------------------------------- ----------
physical read total bytes                                        22327.5781
physical write total bytes                                                0
cell physical IO interconnect bytes                              130.069008
cell physical IO bytes pushed back due to excessive CPU on cell           0
cell physical IO bytes saved during optimized file creation               0
cell physical IO bytes saved during optimized RMAN file restore           0
cell physical IO bytes eligible for predicate offload            22324.6094
cell physical IO bytes saved by storage index                             0
cell physical IO interconnect bytes returned by smart scan       127.100258
cell IO uncompressed bytes                                       22324.6094

EVENT                                    TOTAL_WAITS  WAIT_SECS AVG_WAIT_SECS
---------------------------------------- ----------- ---------- -------------
cell single block physical read                  236        .14         .0006
cell smart table scan                          10880      13.19         .0012
cell multiblock physical read                     17        .02         .0009
```

In this example, the `CUSTOMERS` table has been updated in a way that resulted in row migration across approximately 6.5% of the data blocks in the table. Now, when the query is executed, the query timing, the statistics, and the wait events are close to the original values observed without any migrated rows. However, there is still a noticeable difference between the amount of data returned by Smart Scan and amount of physical interconnect IO. This difference, along with the cell physical read wait events, are symptoms of the row migration present in the `CUSTOMERS` table.



Column Filtering Example

```
SQL> select * from customers;

----------------------------------------------------------------------------
| Id  | Operation                 | Name      | Rows   | Bytes | Cost (%CPU)|
----------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |           |  1239K|   244M| 10242   (1)|
|   1 |  TABLE ACCESS STORAGE FULL| CUSTOMERS |  1239K|   244M| 10242   (1)|
----------------------------------------------------------------------------

NAME                                                                     MB
---------------------------------------------------------------- ----------
physical read total bytes                                        290.335938
cell physical IO interconnect bytes                              290.335938
cell physical IO bytes eligible for predicate offload                     0
cell physical IO interconnect bytes returned by smart scan                0
```

```
SQL> select cust_email from customers;

----------------------------------------------------------------------------
| Id  | Operation                 | Name      | Rows   | Bytes | Cost (%CPU)|
----------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |           |  1239K|    20M| 10235   (1)|
|   1 |  TABLE ACCESS STORAGE FULL| CUSTOMERS |  1239K|    20M| 10235   (1)|
----------------------------------------------------------------------------

NAME                                                                     MB
---------------------------------------------------------------- ----------
physical read total bytes                                        290.289063
cell physical IO interconnect bytes                              29.0223618
cell physical IO bytes eligible for predicate offload            290.289063
cell physical IO interconnect bytes returned by smart scan       29.0223618
```

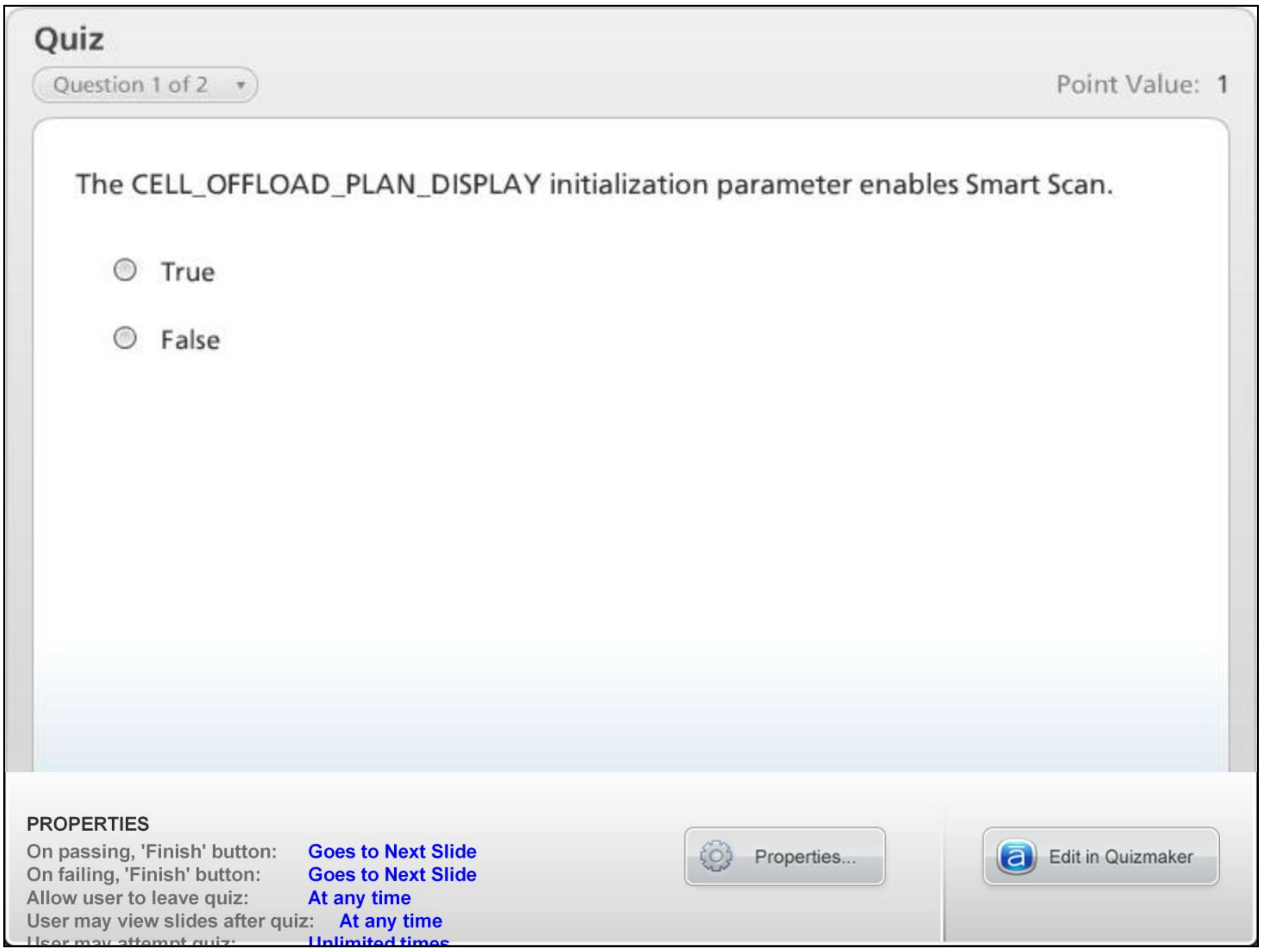
ORACLE

- 26

This example examines the effect of column filtering using two simple SQL queries.

The top query selects the entire customers table. The associated query execution plan shows that the table scan is offloaded to Exadata Storage Server. However, because the query asks for the entire table to be returned the entire table must be transported across the storage network.

The bottom query selects just one column from the `customers` table. Note that the associated query execution plan provides no explicit notification regarding column filtering. It does indicate that the optimizer expects to process a smaller volume of data (20M bytes compared to 244M bytes), which can be used to infer that column filtering will take place. However, the proof of column filtering can be seen from the statistics associated with the query. This time the entire table is eligible for predicate offload but only the data associated with the `cust_email` column is transported across the storage network.


Quiz

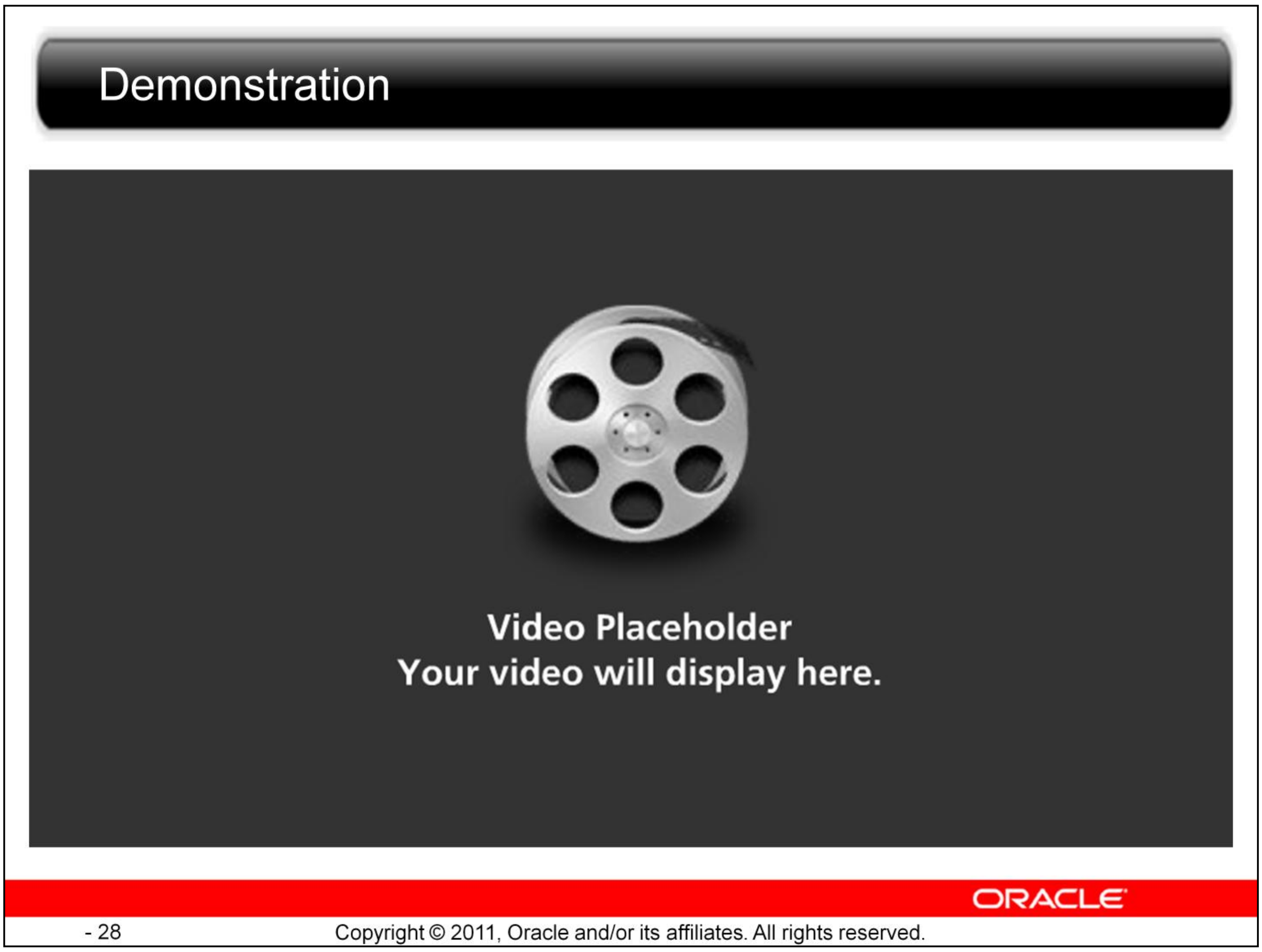
The CELL_OFFLOAD_PLAN_DISPLAY initialization parameter enables Smart Scan.

- True
- False

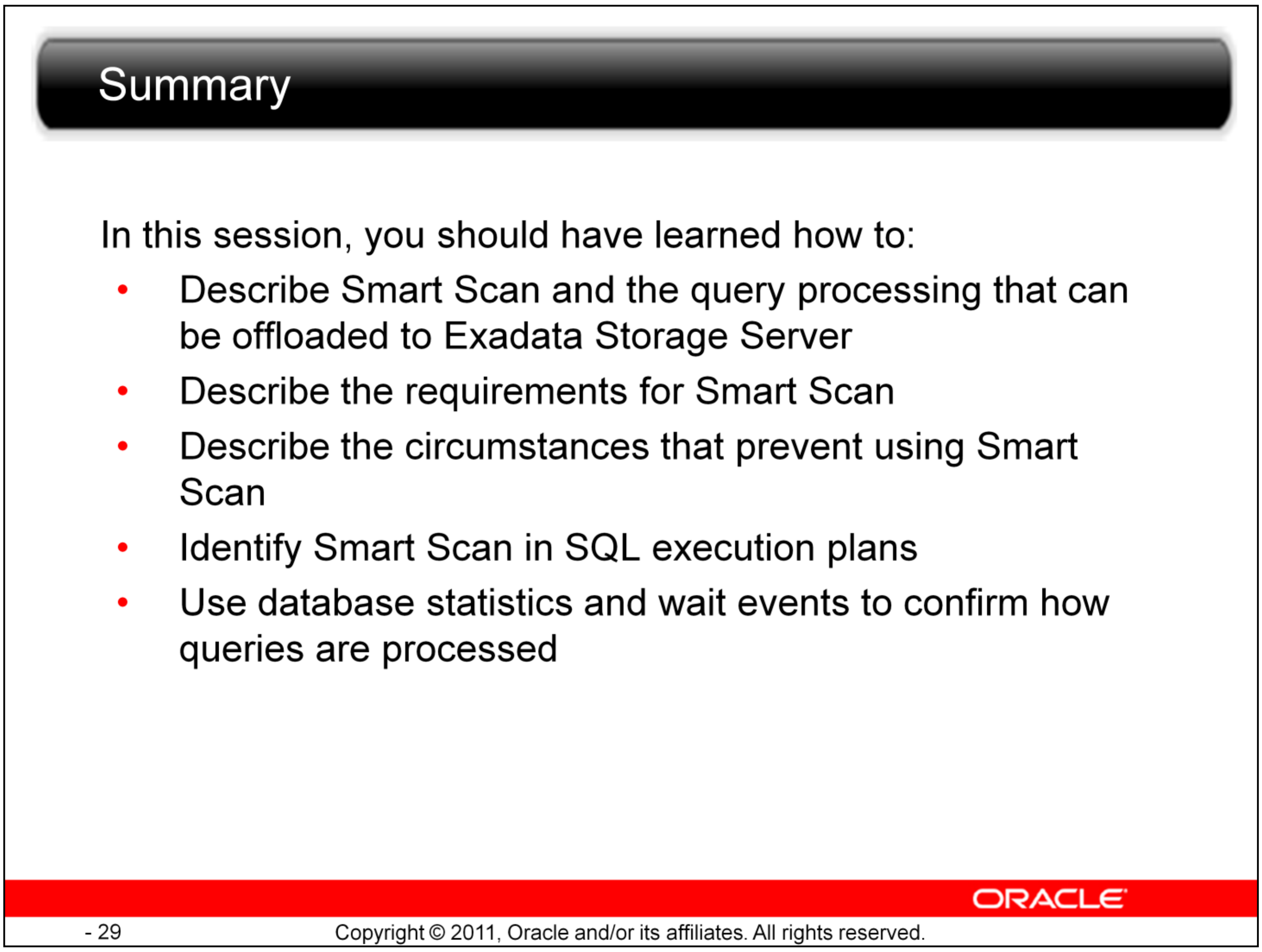
PROPERTIES

On passing, 'Finish' button:	Goes to Next Slide
On failing, 'Finish' button:	Goes to Next Slide
Allow user to leave quiz:	At any time
User may view slides after quiz:	At any time

Now you can test what you've learnt using a couple of quiz questions.



Demonstration



Summary

In this session, you should have learned how to:

- Describe Smart Scan and the query processing that can be offloaded to Exadata Storage Server
- Describe the requirements for Smart Scan
- Describe the circumstances that prevent using Smart Scan
- Identify Smart Scan in SQL execution plans
- Use database statistics and wait events to confirm how queries are processed

- 29