Module 5

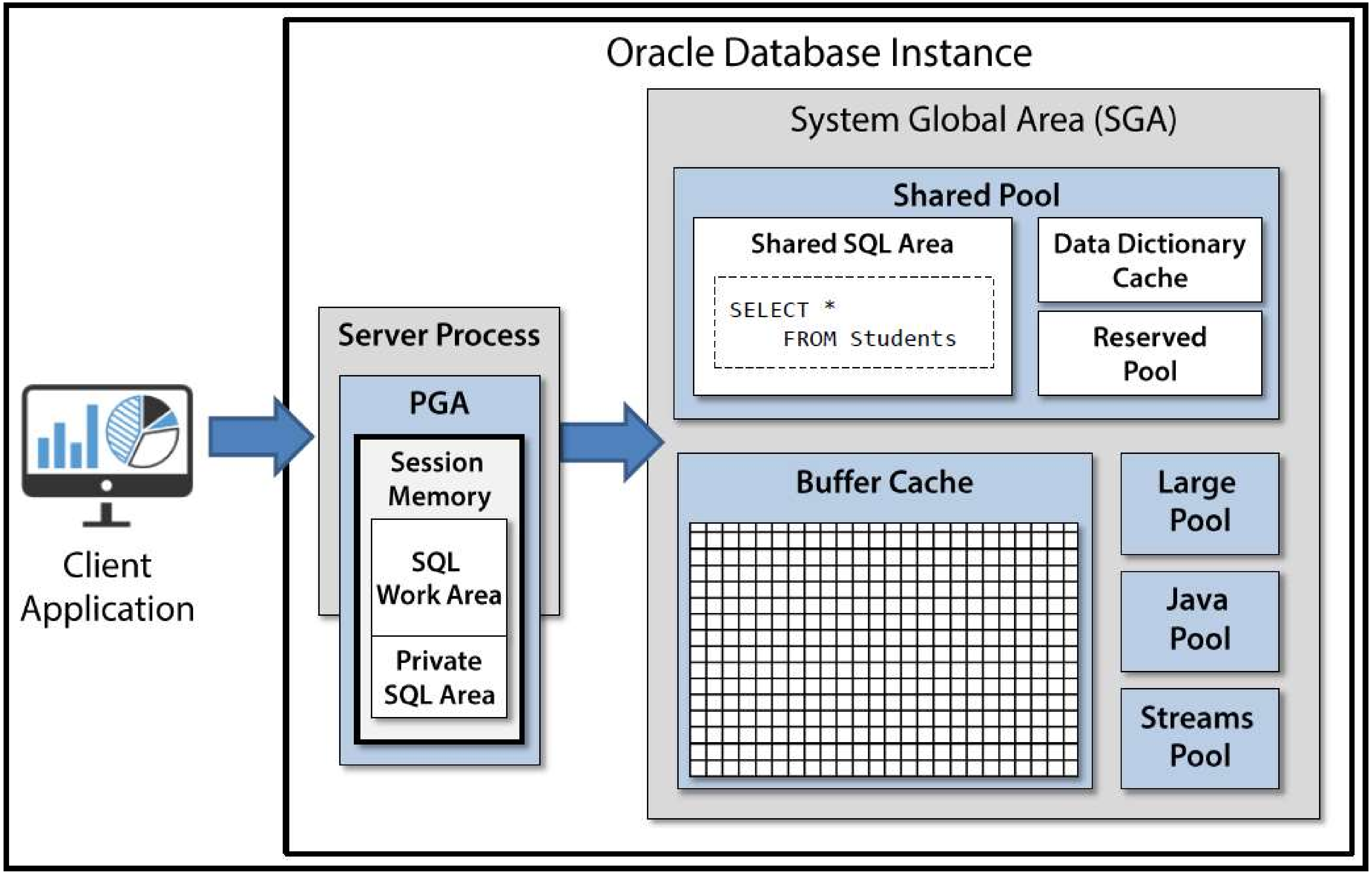
Optmizing Sort Operations

Created by Teerachai Laothong

Agenda

- Oracle Memory Architecture
- Sort Operations
- Sorting and indexing
- Writing top n queries and ranking
- Min/Max Query
- Using Count
- Conditional Group-by
- Avoiding sorting in set operations

Oracle Memory Architecture



Sort Operations

- How Oracle do Sorting
- Single-pass
- Multi-pass
- Optimal sort

How Oracle do Sorting

- If the space needed for sorting is greater than the space reserved for the sort area, the data to be sorted is split into smaller pieces, called sort runs.
- The sort occurs on every single byte, which is stored in temporary segments on-disk.
- The data of sort runs are finally merged together to obtain the final result.

Single-pass, Multi-pass and Optimal Sort

- If there is enough space for this merge operation in the sort area, we have a **single-pass** (on-disk) sort
- Otherwise the merge operation is executed in more steps, in this case, we have a **multi-pass** (on-disk) sort.
- **Optimal sort** is run totally in-memory.

Non-optimal sort

When the I/O operation from and to disk is involved, an `optimal sort` cannot take place, so it is better to have a `single-pass` sort than the `multi-pass`.

PGA_AGGREGATE_TARGET

- We can use the recommended parameter `PGA_AGGREGATE_TARGET` to set the PGA memory available to all server processes attached to the instance.
- Please note that the `PGA_AGGREGATE_TARGET` parameter value can change automatically over time, starting with Oracle Database 11g as part of the Automatic Memory Management enhancements available at 11g.

Sorting and indexing

- We can use indexes to avoid sort operations.

Inherent Sort

If we add an ORDER BY clause in the DISTINCT and GROUP BY queries, there is a change in the execution plans.

```
SELECT
  DISTINCT CUST_CITY
FROM CUSTOMERS
ORDER BY CUST_CITY;

SELECT CUST_CITY, COUNT(*)
FROM CUSTOMERS
GROUP BY CUST_CITY
ORDER BY CUST_CITY;
```

Use Indexing

The index will be not only used to avoid the full table scan operation on the table, but it can also be useful in avoiding the sort operation.

Writing top n queries and ranking

- The correct way to obtains top n rows

The often used statement

```sql
SELECT CUST_ID, CUST_FIRST_NAME, CUST_LAST_NAME, CUST_YEAR_OF_BIRTH
FROM CUSTOMERS
WHERE ROWNUM < 11
ORDER BY CUST_YEAR_OF_BIRTH DESC;
```

The correct statement

```
SELECT * FROM (
  SELECT CUST_ID, CUST_FIRST_NAME, CUST_LAST_NAME, CUST_YEAR_OF_BIRTH
  FROM CUSTOMERS
  ORDER BY CUST_YEAR_OF_BIRTH DESC
)
WHERE ROWNUM < 11;
```

Using RANK()

```sql
SELECT * FROM (
  SELECT CUST_ID, CUST_FIRST_NAME, CUST_LAST_NAME, CUST_YEAR_OF_BIRTH,
  RANK() OVER (ORDER BY CUST_YEAR_OF_BIRTH DESC) AS RANKING
  FROM CUSTOMERS
)
WHERE RANKING < 11;
```

Using DENSE_RANK()

```
SELECT * FROM (
  SELECT CUST_ID, CUST_FIRST_NAME, CUST_LAST_NAME, CUST_YEAR_OF_BIRTH,
  DENSE_RANK() OVER (ORDER BY CUST_YEAR_OF_BIRTH DESC) AS RANKING
  FROM CUSTOMERS
)
WHERE RANKING < 11;
```

Performance

The second query executes in about 175 percent of the time needed by the first query

```
SET TIMING ON
SELECT * FROM (
  SELECT * FROM sh.SALES ORDER BY AMOUNT_SOLD DESC
) WHERE ROWNUM < 11;

SELECT * FROM (
  SELECT S.*,
    DENSE_RANK() OVER (ORDER BY AMOUNT_SOLD DESC) AS RANKING
    FROM sh.SALES S
) WHERE RANKING < 11;
SET TIMING OFF
```

Min/Max Query

Typical Min/Max Query

```
SELECT MAX(CUST_CREDIT_LIMIT) FROM CUSTOMERS;
SELECT MIN(CUST_CREDIT_LIMIT) FROM CUSTOMERS;
```

Query both Min and Max

```
SELECT MAX(CUST_CREDIT_LIMIT), MIN(CUST_CREDIT_LIMIT)
FROM CUSTOMERS;
```

The better way

```sql
SELECT
  MIN(CUST_CREDIT_LIMIT) AS MIN_VALUE,
  0 AS MAX_VALUE
FROM CUSTOMERS
UNION ALL
SELECT
  0,
  MAX(CUST_CREDIT_LIMIT)
FROM CUSTOMERS;
```

Use Indexing

Indexing help improve performance for those queries.

Using Count

Two popular counts

```
SELECT COUNT(*) FROM CUSTOMERS;
SELECT COUNT(1) FROM CUSTOMERS;
```

They are the same query.

Use Indexing

Indexing help improve performance for those queries.

Conditional Group-by

A group-by query with a filter

```sql
SELECT CUST_CREDIT_LIMIT, MAX(CUST_YEAR_OF_BIRTH) AS DATAMAX
FROM CUSTOMERS
GROUP BY CUST_CREDIT_LIMIT
HAVING CUST_CREDIT_LIMIT > 10000
ORDER BY CUST_CREDIT_LIMIT;
```

A group-by query with the same filter

```
SELECT CUST_CREDIT_LIMIT, MAX(CUST_YEAR_OF_BIRTH) AS DATAMAX
FROM CUSTOMERS
WHERE CUST_CREDIT_LIMIT > 10000
GROUP BY CUST_CREDIT_LIMIT
ORDER BY CUST_CREDIT_LIMIT;
```

Use Indexing

We can use the bitmap index instead of a full table scan of the table.

Avoiding sorting in set operations

- INTERSECT and JOIN
- MINUS and ANTI-JOIN

INTERSECT and JOIN

Using the `INTERSECT` operator

```
SELECT CUST_LAST_NAME AS LastName FROM sh.CUSTOMERS
INTERSECT
SELECT LAST_NAME FROM sh.MY_EMPLOYEES;
```

Using the `JOIN` operator

```
SELECT DISTINCT
  C.CUST_LAST_NAME AS LastName
FROM sh.CUSTOMERS C
  INNER JOIN sh.MY_EMPLOYEES E
    ON C.CUST_LAST_NAME = E.LAST_NAME;
```

The latter is perform better

MINUS and ANTI-JOIN

Using `MINUS` operator

```
SELECT C.CUST_LAST_NAME AS LastName FROM sh.CUSTOMERS C
MINUS
SELECT E.LAST_NAME FROM sh.MY_EMPLOYEES E;
```

Using `ANTI-JOIN` operator

```
SELECT DISTINCT
  C.CUST_LAST_NAME AS LastName
FROM sh.CUSTOMERS C
WHERE C.CUST_LAST_NAME NOT IN (
  SELECT E.LAST_NAME FROM sh.MY_EMPLOYEES E);
```

The latter is perform better

THE END

- Source code & documentation
- Back to Course Outline