School of Computing Technologies

ISYS1101/ 1102 Database Applications

Week 5: Tute/Lab – Database Partitioning

Semester 2 2023

1 Objective

The objective of this tute/lab session is to learn more about table partitioning and continue hands-on partitioning exercises with Oracle. The activities in this tute/lab session is based on the material covered in Week 4 lecture.

In particular, you will:

- Learn about different partitioning strategies;
- Apply SQL DDL statements to create partitions and maintain them;
- Using partition pruning, partition-wise joins, and parallel DML on partitioned tables to optimise query performance.

2 Preparation Tasks

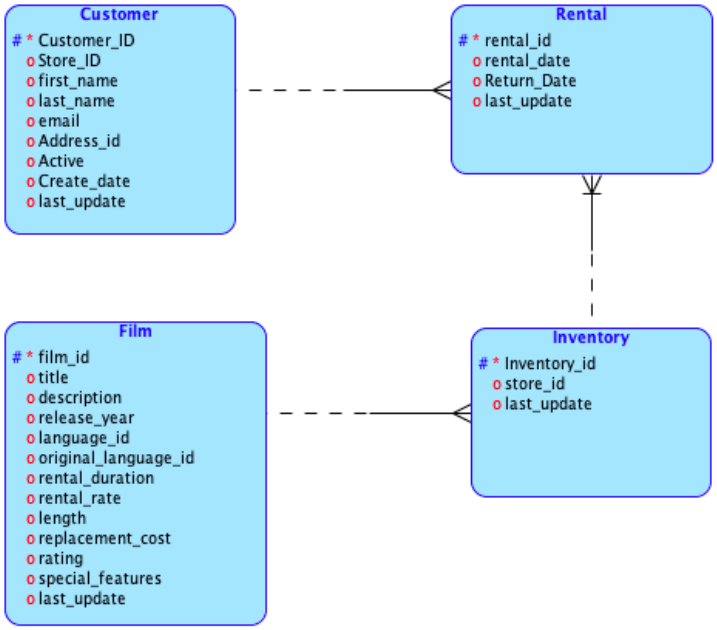
From now on, you cannot use sample databases. You must have your own database schema. Contact ITS if you still did not receive an email on connection settings.

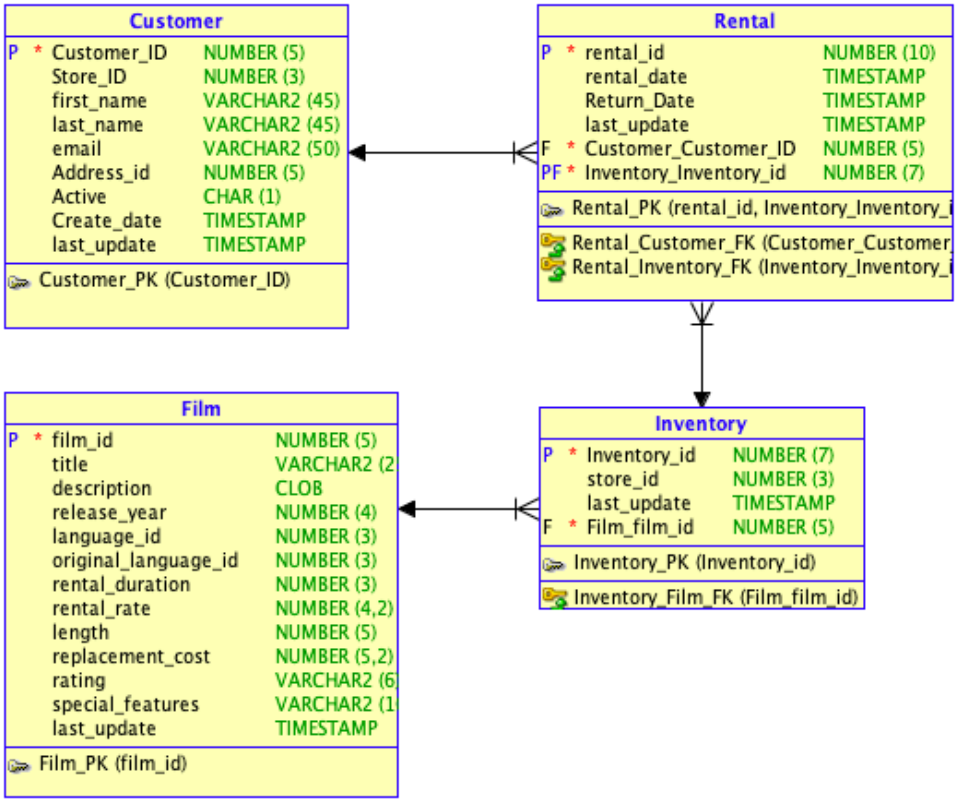
Note: If you have already build the Movie Rental Database as part of the Week 4 lab work, you are not required to repeat it here.

Last week we downloaded and built the rentals table using `rental-DDL.sql` and `rental-populate-bulk-ALL.sql`. This table is a part of a movie rentals database (extracted from MySQL sample database – Sakila movie rental database). For the activities in this week's tute/ lab session, you are required to download DDL and populate scripts for the following tables in this database.

- Customer
- Film
- Inventory

Download `<tablename>-DDL.sql` and `<tablename>-populate-bulk-ALL.sql` scripts from Canvas (Modules → Course Resources → Movie Rentals Database). Run them on SQL Developer to build these four tables.
The underlying data model (conceptual and physical) that these tables are built on is as follows:





Please note that the tables do not have any constraints (no primary keys or foreign keys) built on to them. That's intentional, as we explore the impact of indexes and add them later.

Your Lab assistant will discuss with you the performance of simple INSERTs and bulk INSERTs, especially you upload thousands of rows at once.

3 Tutorial Questions on Query Evaluation

3.1 Task 1: Partition a table using List Partition

In our "film" table, the films are given a classification rating by the authorities (https://www.classification.gov.au/classification-ratings/latest-classification-decisions). It was found that in our film collection, the films are classified as follows:

Rating	Count
PG-13	223
R	195
NC-17	210
G	178
PG	194

As the table grows with more and more films added, it is recommended that it must be partitioned to improve query performance. Furthermore, it was observed that bulk of the queries on this table involve the 'rating' column as well.

Activity 1: Rebuild the 'film' table using list partition with 'rating' column as the partition key. You must given it a new name (say, 'film_list_partitioned')

Keep the original 'film' table, as it is useful to compare the performance of queries on both un-partitioned and partitioned tables.
Hint: You may not see any substantial cost savings because the tables are relatively small.

Activity 2: Re-populate the new table using the same sat of data. You may use a slightly modified version of the bulk-populate file (film-populate-bulk-all.SQL) you have used in Lab 4.

Activity 3: Check the table partition structure. Oracle allows you to view the data dictionary, which stores all relevant meta data about tables and other related database objects. Use the following SQL statement.

```
SELECT partition_name, tablespace_name
    FROM user_tab_partitions
    WHERE table_name = upper('film_list_partitioned');
```

3.2 Query a partitioned table

Activity 4: Run the following SQL query and extract the Explain Plan output.

```
SELECT title, rating
    FROM film
    WHERE rating = 'PG';
```

Run the same query, on the partitioned table. Extract the Explain Plan output.

```
SELECT title, rating
    FROM film_list_partitioned
    WHERE rating = 'PG';
```

What are the differences you observed? Discuss with your group members on the benefits of the partitioning in this particular example.

Activity 5: Run the following SQL query and extract the Explain Plan output.

```
SELECT title, rating
    FROM film
    WHERE length > 60;
```

Run the same query, on the partitioned table. Extract the Explain Plan output.

```
SELECT title, rating
    FROM film_list_partitioned
    WHERE length > 60;
```

What are the differences you observed? Discuss with your group members on the benefits of the partitioning in this particular example.

3.3 Adding a new partition to an already partitioned table

Let's suppose a new classification rating, say, MA15+, has been introduced to movie classification in Australia. Oracle allows two ways to add partitions:

1. Add a partition to the end of the partition list;
2. Split a current partition and re-distribute data

Activity 6: Run the following query:

```
ALTER TABLE film_list_partitioned
   ADD PARTITION film_MA15plus VALUES ('MA15+');
```

Were you able to run the query?

If it failed, why did it fail?

How do you overcome this problem?

3.4 Retrieving Data from a chosen partition

Normally, when you state the data source in an SQL query, you normally state the base table. The query optimiser determines the partitions it must use to source required data. However, if you know the partition key and your query is based on partition key, you can eliminate the query optimiser's role and you can explicitly run a query to extract data from a chosen partition.

Activity 7: Write a query to list films classified as 'PG' using the PG partition.

```
SELECT title, rating
    FROM film_list_partitioned PARTITION (film_PG)
    WHERE rating = 'PG';
```

Now, check the execution plan.

Do you see a difference between this execution plan and the execution plan you got in Activity 4 above?

Why?

3.5 Splitting Partitions

As tables grow in size, sometimes a need arises to splitting existing partitions. For example, originally movies in both 'PG' and 'PG-13' could have been stored in one partition, however, due to the growth we may require splitting of this partition.

Activity 8: Write a ALTER TABLE statement to split 'PG' partition to make two partitions.

```
ALTER TABLE film_list_partitioned
   SPLIT PARTITION pg VALUES ('PG-13')
   INTO
    ( PARTITION pg-13,
      PARTITION pg
    )
```

Observe how the explicitly specified partition key values ('PG-13' in this example) pushed to the newly-created first partition and the remaining values are kept in the old partition.

3.6 Range Partitioning

In our IMDB database, it is possible to partition 'film' table using range partitioning.

Activity 9: Do this as a small group:

1. What are the benefits of doing range partition?
2. Identify a few queries that can benefit from range partitioning.
3. Identify a few queries that does not receive any support from range partitioning.

Activity 10: Rebuild the 'film' table using list partition with 'release_year' column as the partition key. You must given it a new name (say, 'film_range_partitioned')

Keep the original 'film' table, as it is useful to compare the performance of queries on both un-partitioned and partitioned tables.
Hint: You may not see any substantial cost savings because the tables are relatively small.

Activity 11: Re-populate the new table using the same sat of data. You may use a slightly modified version of the bulk-populate file (film-populate-bulk-all.SQL) you have used in Lab 4.

Activity 12: Check the table partition structure. Oracle allows you to view the data dictionary, which stores all relevant meta data about tables and other related database objects. Use the following SQL statement.

```
SELECT partition_name, tablespace_name
    FROM user_tab_partitions
    WHERE table_name = upper('film_range_partitioned');
```

3.7 Hash Partitioning

In our IMDB database, it is possible to partition 'film' table using hash partitioning.

Activity 13: Do this as a small group:

4. What are the benefits of doing hash partition?
5. Identify a few queries that can benefit from hash partitioning.
6. Identify a few queries that does not receive any support from hash partitioning.

Activity 14: Rebuild the 'film' table using list partition with 'title column as the partition key. You must given it a new name (say, 'film_hash_partitioned')

Keep the original 'film' table, as it is useful to compare the performance of queries on both un-partitioned and partitioned tables.
Hint: You may not see any substantial cost savings because the tables are relatively small.

Activity 15: Re-populate the new table using the same sat of data. You may use a slightly modified version of the bulk-populate file (film-populate-bulk-all.SQL) you have used in Lab 4.

Activity 16: Check the table partition structure. Oracle allows you to view the data dictionary, which stores all relevant meta data about tables and other related database objects. Use the following SQL statement.

```
SELECT partition_name, tablespace_name
    FROM user_tab_partitions
    WHERE table_name = upper('film_hash_partitioned');
```