To optimize this query, the key areas to focus on include indexing, minimizing subquery executions, and ensuring efficient joins. Here's a breakdown:

Query Breakdown

Inner Query (rooms2):

Fetches room\_id where exactly 1 guest has booked a room (count(guest\_id) = 1) between the given checkin and checkout dates.

Outer Query (rooms1):

Joins rooms1 with the result of the inner query (rooms2) on room\_id to fetch details of rooms that meet the criteria.

Optimization Steps

Indexes

Indexes are critical for performance, especially when dealing with joins and filtering. For this query, consider creating the following indexes:

bookings(checkin, checkout): This index will speed up the filtering on the checkin and checkout columns.

*CREATE INDEX idx\_checkin\_checkout ON hotelbookingdb.bookings(checkin, checkout);*

bookings(room\_id, guest\_id): As you’re grouping by room\_id and counting guest\_id, this composite index will help reduce the time required for the group and count operations.

*CREATE INDEX idx\_room\_guest ON hotelbookingdb.bookings(room\_id, guest\_id);*

rooms(id): Ensure that [rooms.id](http://rooms.id/) is indexed, as it’s used in the join condition.

*CREATE INDEX idx\_room\_id ON hotelbookingdb.rooms(id);*

Join Strategy

Ensure the [rooms.id](http://rooms.id/) column is indexed for efficient joining. Since you are performing an INNER JOIN, this ensures that only rooms matching the subquery results are fetched, reducing the data processed.

Check Execution Plan

After applying the indexes, you should check the query execution plan (EXPLAIN in SQL) to ensure that the query optimizer is using the indexes correctly. This will help identify if further optimizations are necessary, such as rewriting joins or reducing scanned rows.

By applying proper indexing and reviewing the execution plan, the performance of the query should improve significantly.

