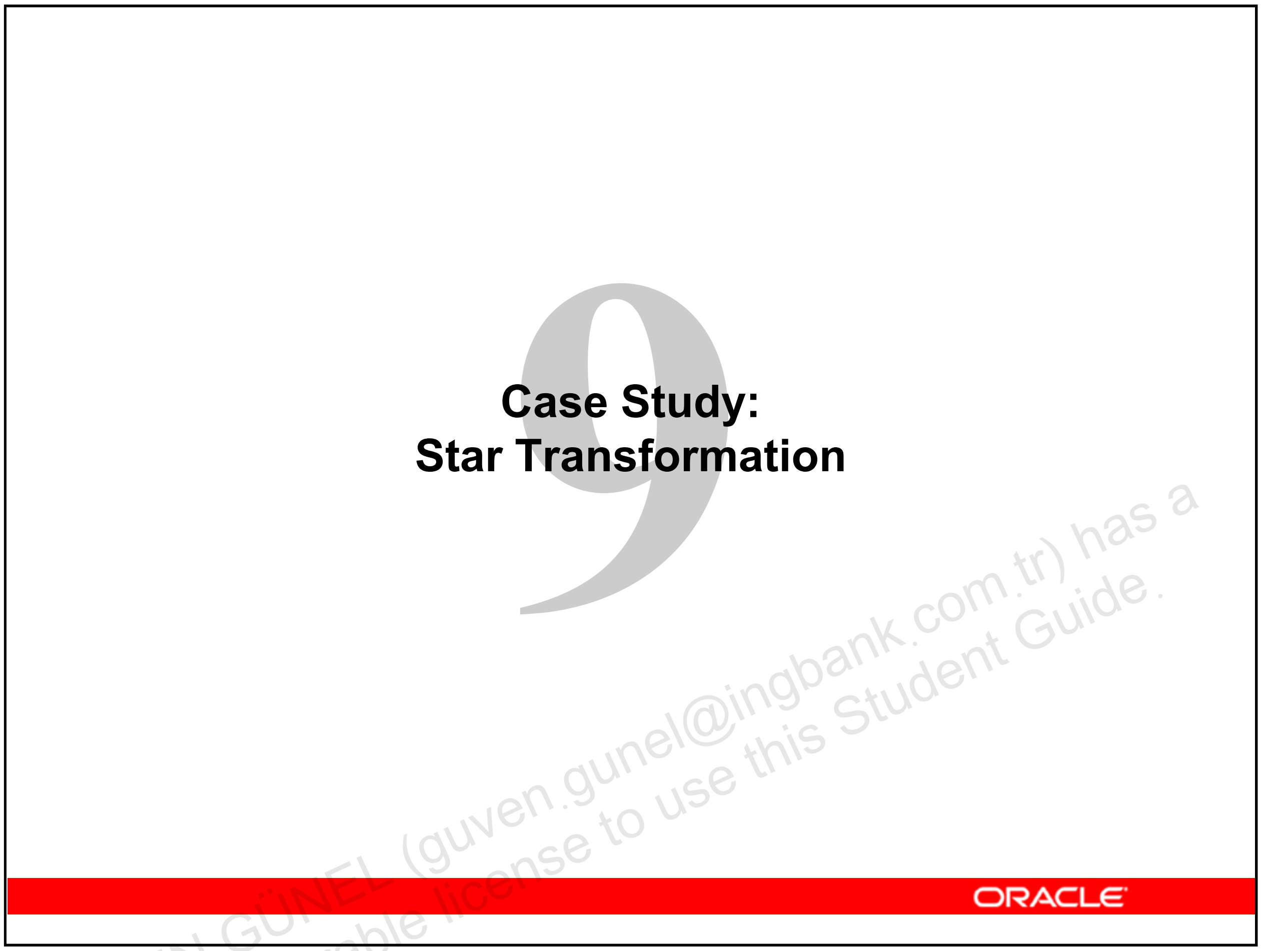
Case Study: Star Transformation

Chapter 9

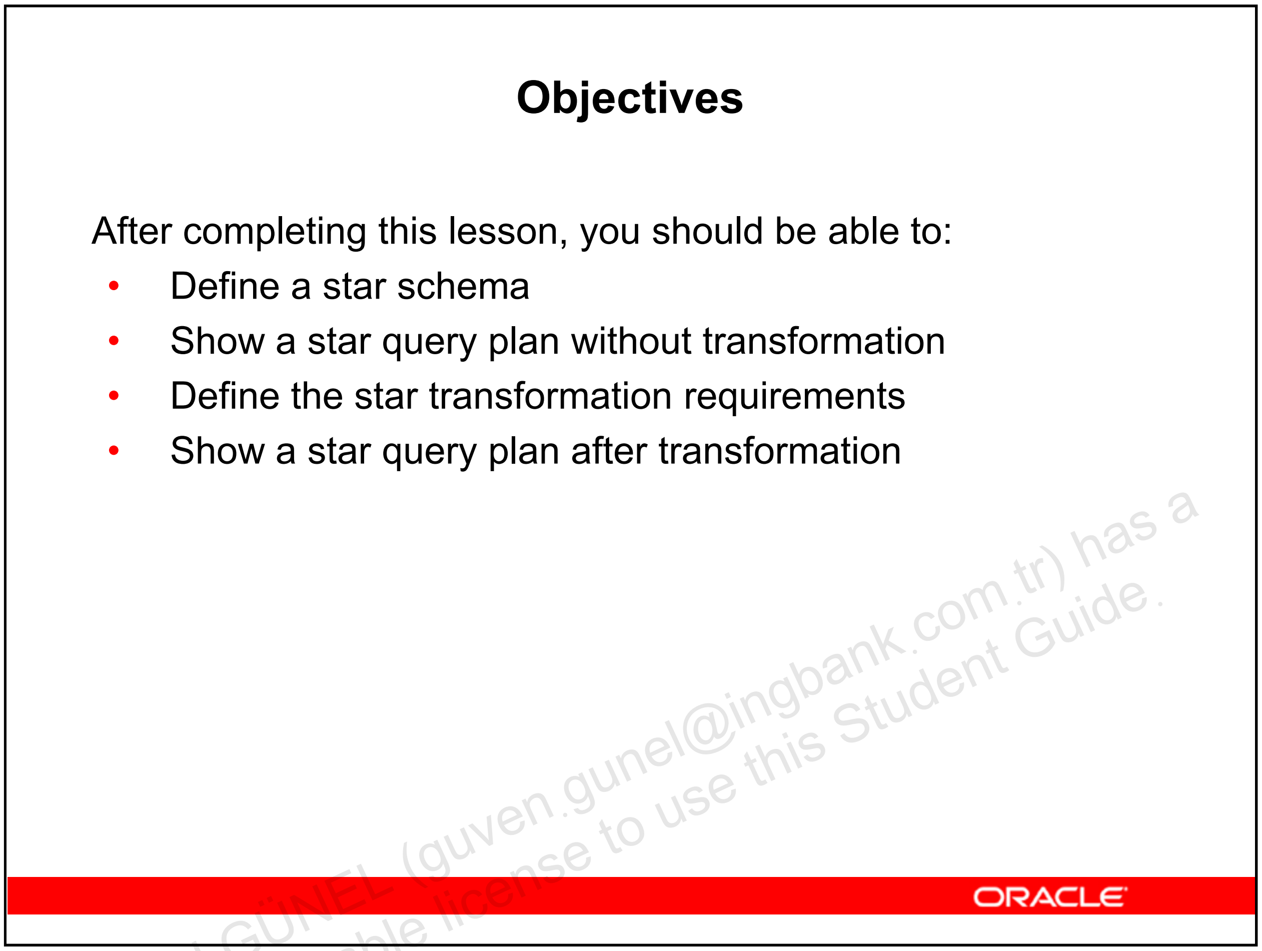


9

Case Study: Star Transformation

ORACLE

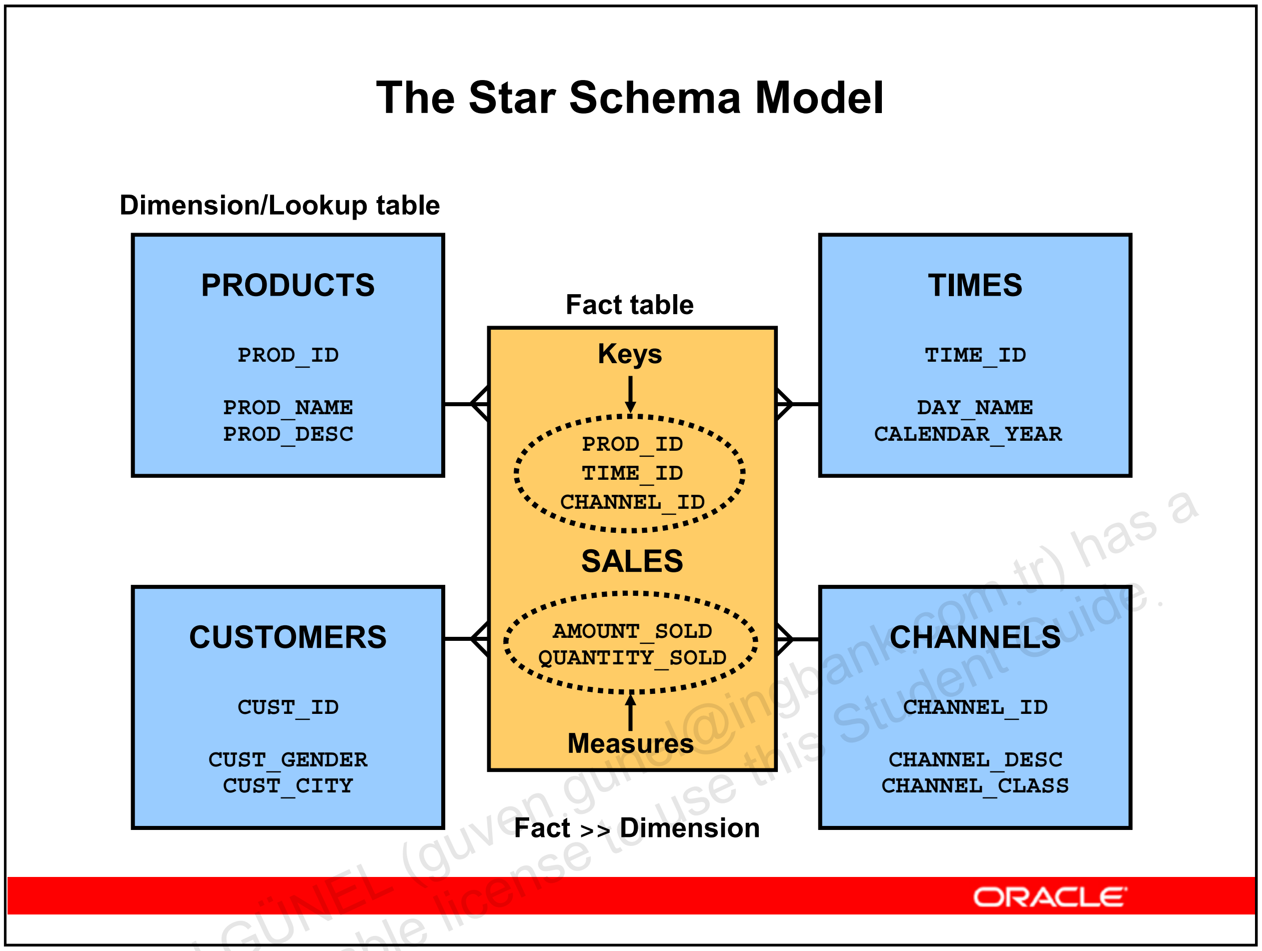
Objectives



Objectives

After completing this lesson, you should be able to:

- Define a star schema
- Show a star query plan without transformation
- Define the star transformation requirements
- Show a star query plan after transformation



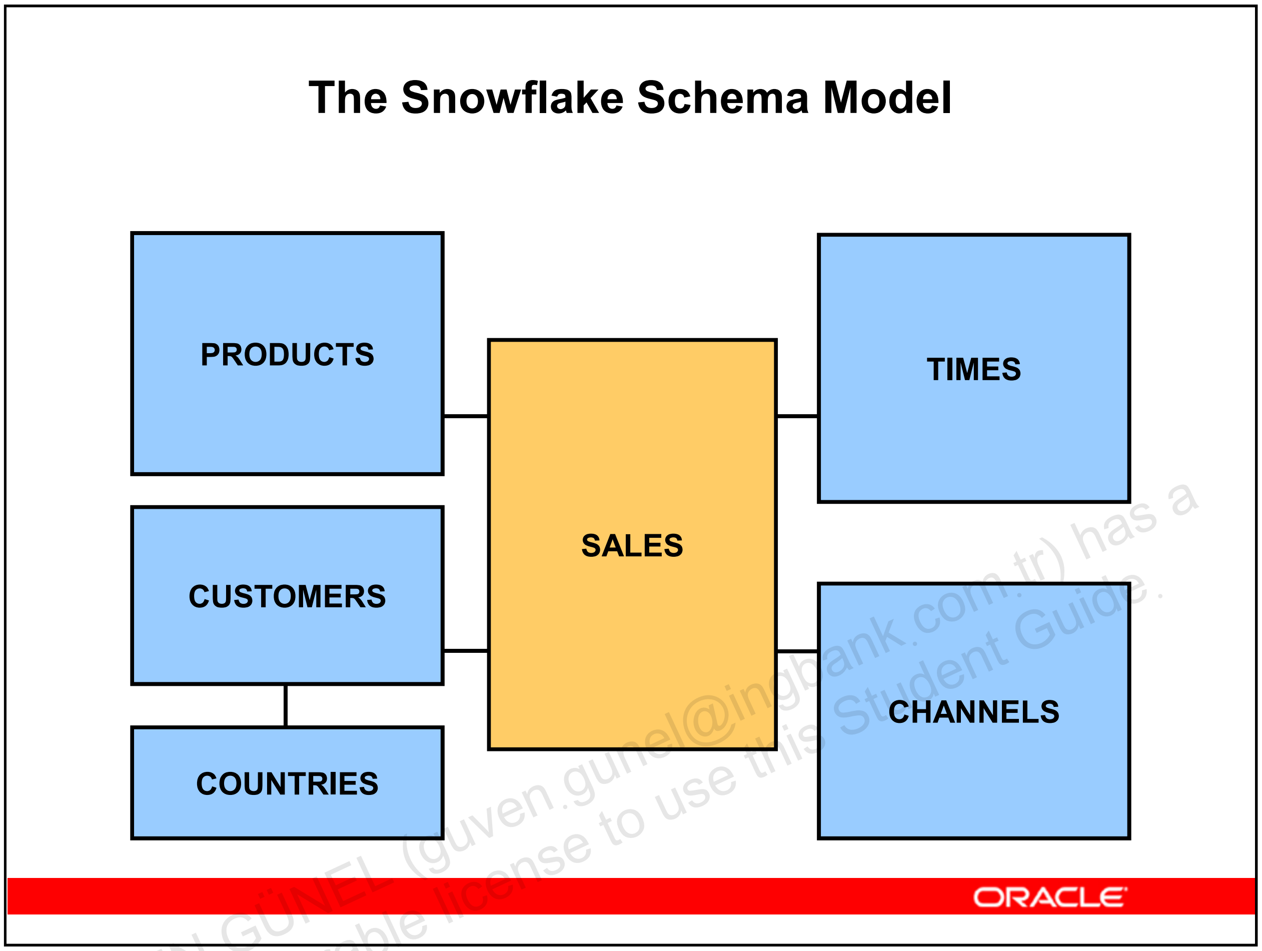
The Star Schema Model

The Star Schema Model

The star schema is the simplest data warehouse schema. It is called a star schema because the entity-relationship diagram of this schema resembles a star, with points radiating from a central table. The center of the star consists of one or more fact tables and the points of the star are the dimension tables. A star schema is characterized by one or more very large fact tables that contain the primary information in the data warehouse and a number of much smaller dimension tables (or lookup tables), each of which contains information about the entries for a particular attribute in the fact table. A star query is a join between a fact table and a number of dimension tables. Each dimension table is joined to the fact table using a primary key to foreign key join, but the dimension tables are not joined to each other. The cost-based optimizer (CBO) recognizes star queries and generates efficient execution plans for them. A typical fact table contains keys and measures. For example, in the Sales History schema, the `sales` fact table contains the `quantity_sold`, `amount`, and `cost` measures, and the `cust_id`, `time_id`, `prod_id`, `channel_id`, and `promo_id` keys. The dimension tables are `customers`, `times`, `products`, `channels`, and `promotions`. The `products` dimension table, for example, contains information about each product number that appears in the fact table.

Note: It is easy to generalize this model to include more than one fact table.

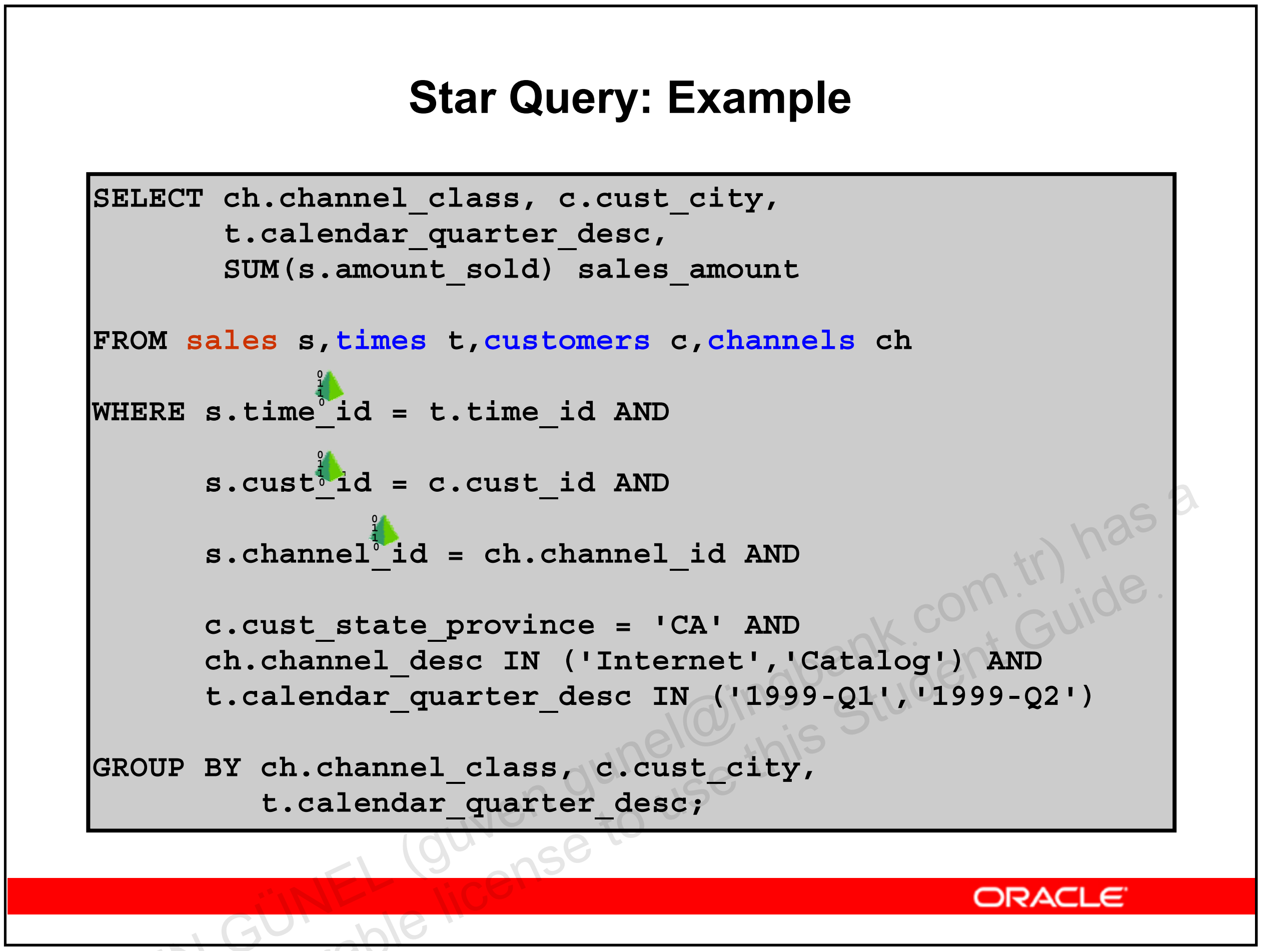
The Snowflake Schema Model


The Snowflake Schema Model

The Snowflake Schema Model

The snowflake schema is a more complex data warehouse model than a star schema, and is a type of star schema. It is called a snowflake schema because the diagram of the schema resembles a snowflake. Snowflake schemas normalize dimensions to eliminate redundancy. That is, the dimension data has been grouped into multiple tables instead of one large table. For example, a product dimension table in a star schema might be normalized into a `products` table, a `product_category` table, and a `product_manufacturer` table in a snowflake schema or, as shown in the slide, you can normalize the `customers` table using the `countries` table. While this saves space, it increases the number of dimension tables and requires more foreign key joins. The result is more complex queries and reduced query performance.

Note: It is suggested that you select a star schema over a snowflake schema unless you have a clear reason to choice the snowflake schema.



Star Query: Example

```
SELECT ch.channel_class, c.cust_city,
       t.calendar_quarter_desc,
       SUM(s.amount_sold) sales_amount

FROM sales s,times t,customers c,channels ch

WHERE s.time_id = t.time_id AND

      s.cust_id = c.cust_id AND

      s.channel_id = ch.channel_id AND

      c.cust_state_province = 'CA' AND
      ch.channel_desc IN ('Internet','Catalog') AND
      t.calendar_quarter_desc IN ('1999-Q1','1999-Q2')

GROUP BY ch.channel_class, c.cust_city,
         t.calendar_quarter_desc;
```

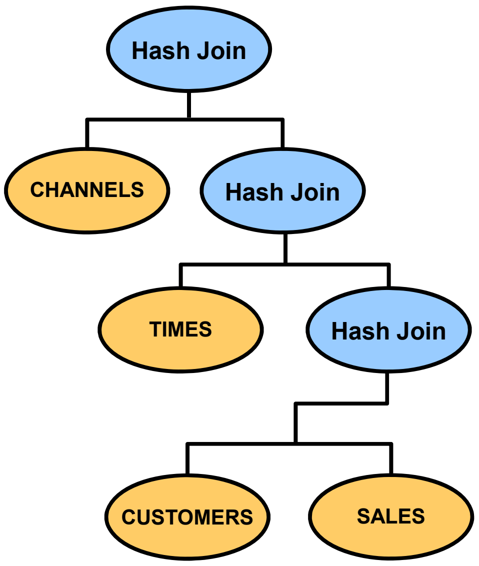
ORACLE

Star Query: Example

Consider the star query in the slide. For the star transformation to operate, it is supposed that the `sales` table of the Sales History schema has bitmap indexes on the `time_id`, `channel_id`, and `cust_id` columns.

Execution Plan Without Star Transformation

Execution Plan Without Star Transformation



```
---------------------------------------------------
| Id  | Operation                | Name           |
---------------------------------------------------
|   0 | SELECT STATEMENT         |                |
|   1 |  HASH GROUP BY           |                |
|*  2 |   HASH JOIN              |                |
|*  3 |    TABLE ACCESS FULL     | CHANNELS       |
|*  4 |    HASH JOIN             |                |
|*  5 |     TABLE ACCESS FULL    | TIMES          |
|*  6 |     HASH JOIN            |                |
|*  7 |      TABLE ACCESS FULL| CUSTOMERS      |
|   8 |      TABLE ACCESS FULL| SALES          |
---------------------------------------------------
```

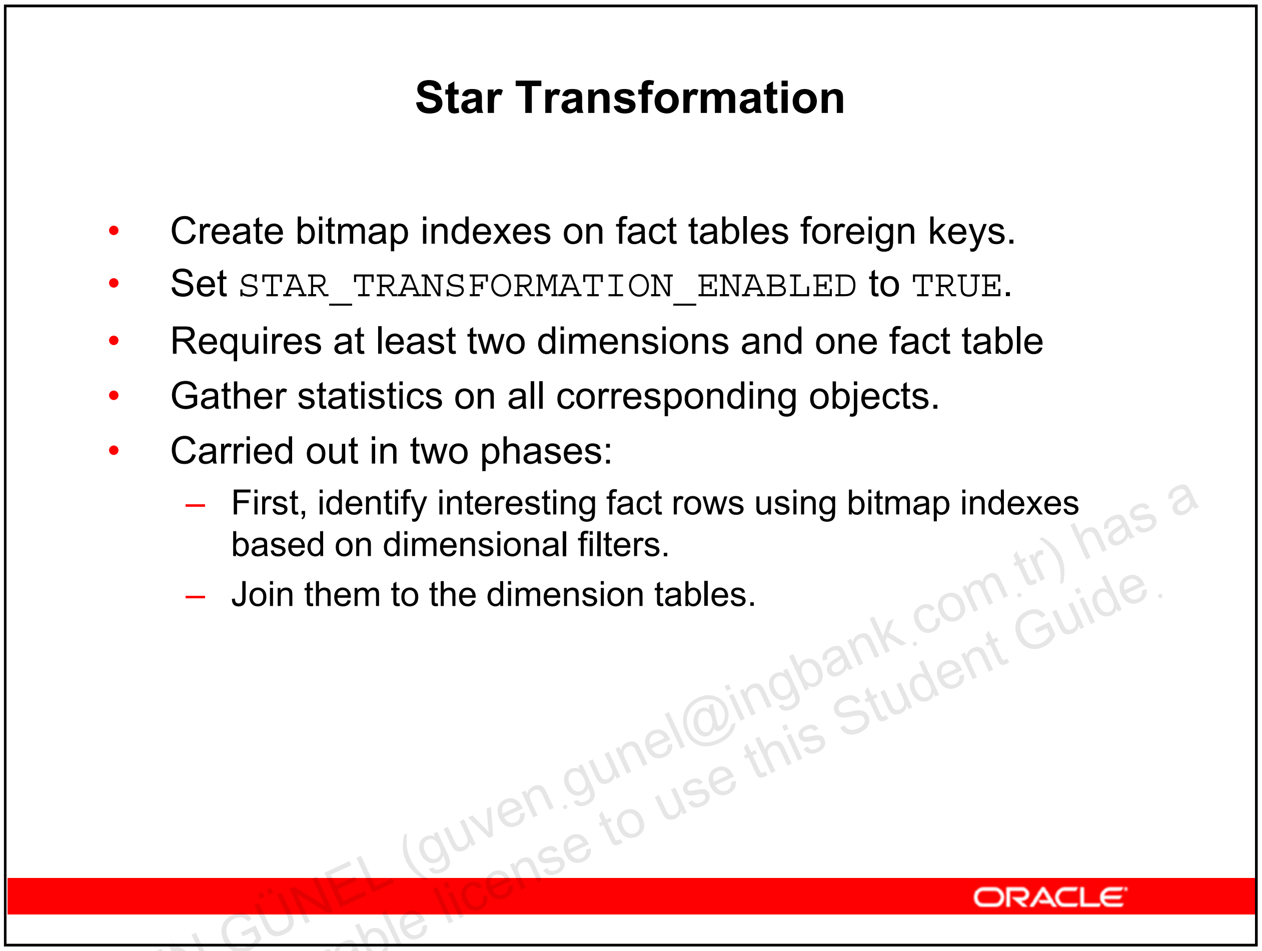
```
Predicate Information (by operation id):
---------------------------------------------
2 - access("S"."CHANNEL_ID"="CH"."CHANNEL_ID")
3 - filter("CH"."CHANNEL_DESC"='Catalog' OR
       "CH"."CHANNEL_DESC"='Internet')
4 - access("S"."TIME_ID"="T"."TIME_ID")
5 - filter("T"."CALENDAR_QUARTER_DESC"='1999-Q1'
    OR "T"."CALENDAR_QUARTER_DESC"='1999-Q2')
6 - access("S"."CUST_ID"="C"."CUST_ID")
7 - filter("C"."CUST_STATE_PROVINCE"='CA')
```

ORACLE

Execution Plan Without Star Transformation

Before you see the benefits of a star transformation, you should review how a join on a star schema is processed without their benefits.

The fundamental issue with the plan in the slide is that the query always starts joining the SALES table to a dimension table. This results in a very large number of rows that can only be trimmed down by the other parent joins in the execution plan.



Star Transformation

- Create bitmap indexes on fact tables foreign keys.
- Set STAR_TRANSFORMATION_ENABLED to TRUE.
- Requires at least two dimensions and one fact table
- Gather statistics on all corresponding objects.
- Carried out in two phases:
  - First, identify interesting fact rows using bitmap indexes based on dimensional filters.
  - Join them to the dimension tables.

Star Transformation

To get the best possible performance for star queries, it is important to follow some basic guidelines:

- A bitmap index should be built on each of the foreign key columns of the fact table or tables.
- The `STAR_TRANSFORMATION_ENABLED` initialization parameter should be set to `TRUE`. This enables an important optimizer feature for star queries. It is set to `FALSE` by default for backwards compatibility.

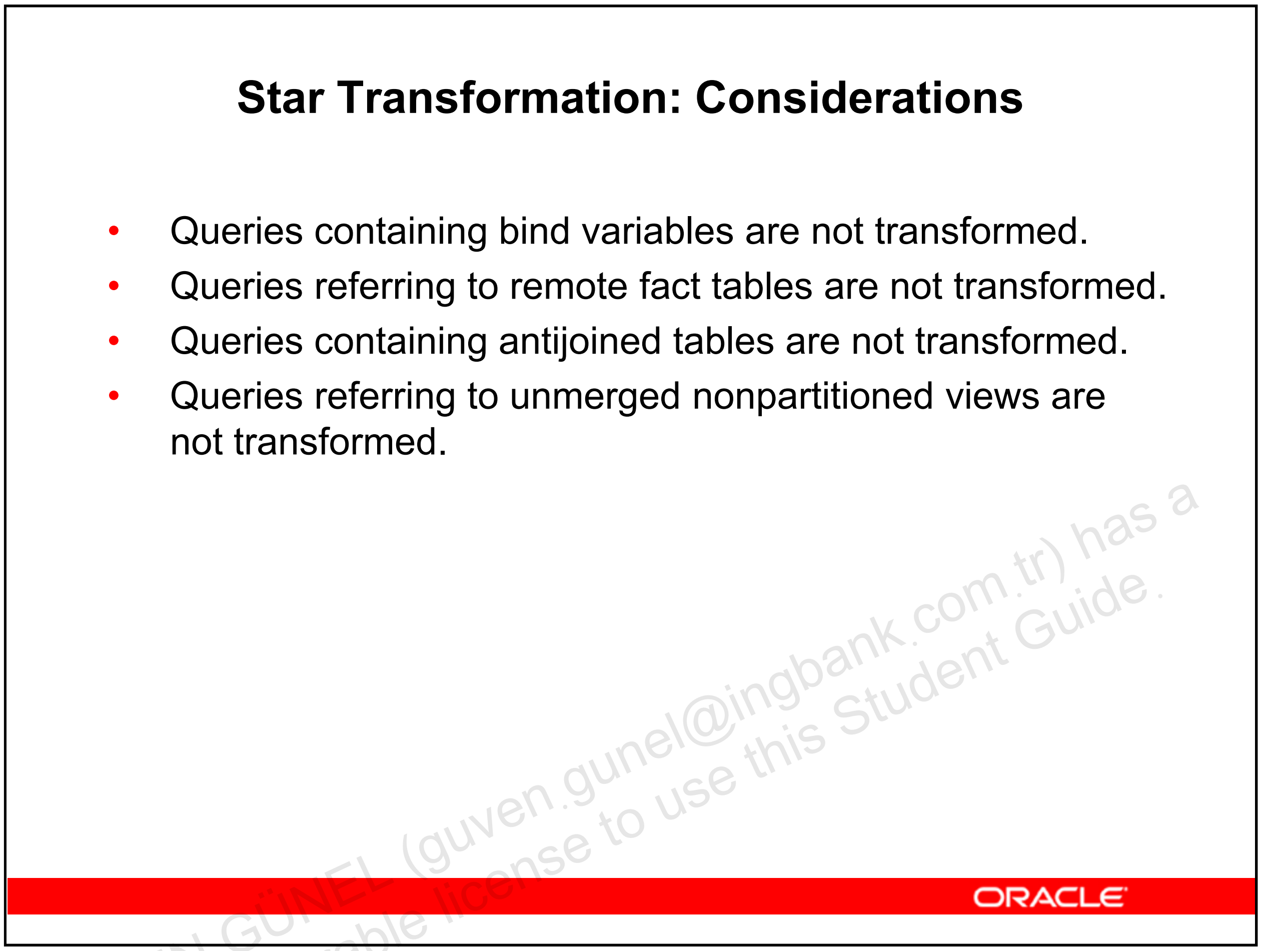
When a data warehouse satisfies these conditions, the majority of the star queries that run in the data warehouse use a query execution strategy known as star transformation. Star transformation provides very efficient query performance for star queries.

Star transformation is a powerful optimization technique that relies on implicitly rewriting (or transforming) the SQL of the original star query. The end user never needs to know any of the details about the star transformation. The system's CBO automatically selects star transformation where appropriate. The optimizer creates an execution plan that processes a star query using two basic phases:

- The first phase retrieves exactly the necessary rows from the fact table (the result set). Because this retrieval utilizes bitmap indexes, it is very efficient.

- The second phase joins this result set to the dimension tables. This operation is called a semijoin.

Note: At least three tables are used in the query (two dimensions and one fact).



Star Transformation: Considerations

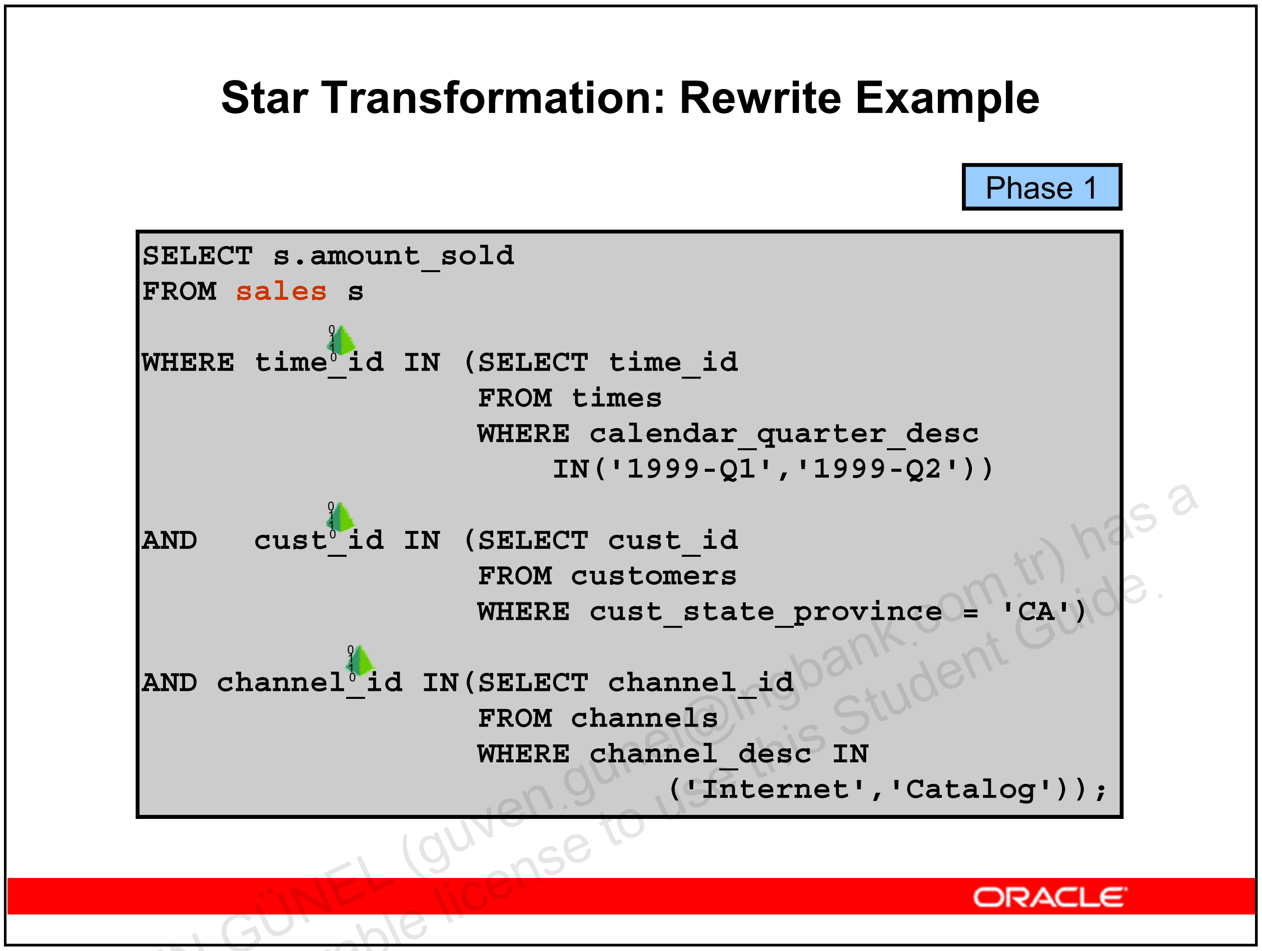
- Queries containing bind variables are not transformed.
- Queries referring to remote fact tables are not transformed.
- Queries containing antijoined tables are not transformed.
- Queries referring to unmerged nonpartitioned views are not transformed.

ORACLE

Star Transformation: Considerations

Star transformation is not supported for tables with any of the following characteristics:

- Queries with a table hint that is incompatible with a bitmap access path
- Queries that contain bind variables
- Tables with too few bitmap indexes. There must be a bitmap index on a fact table column for the optimizer to generate a subquery for it.
- Remote fact tables. However, remote dimension tables are allowed in the subqueries that are generated.
- Antijoined tables
- Tables that are already used as a dimension table in a subquery
- Tables that are really unmerged views, which are not view partitions



Star Transformation: Rewrite Example

Phase 1

```
SELECT s.amount_sold
FROM sales s

WHERE time_id IN (SELECT time_id
                  FROM times
                  WHERE calendar_quarter_desc
                      IN('1999-Q1','1999-Q2'))

AND   cust_id IN (SELECT cust_id
                  FROM customers
                  WHERE cust_state_province = 'CA')

AND channel_id IN(SELECT channel_id
                  FROM channels
                  WHERE channel_desc IN
                       ('Internet','Catalog'));
```

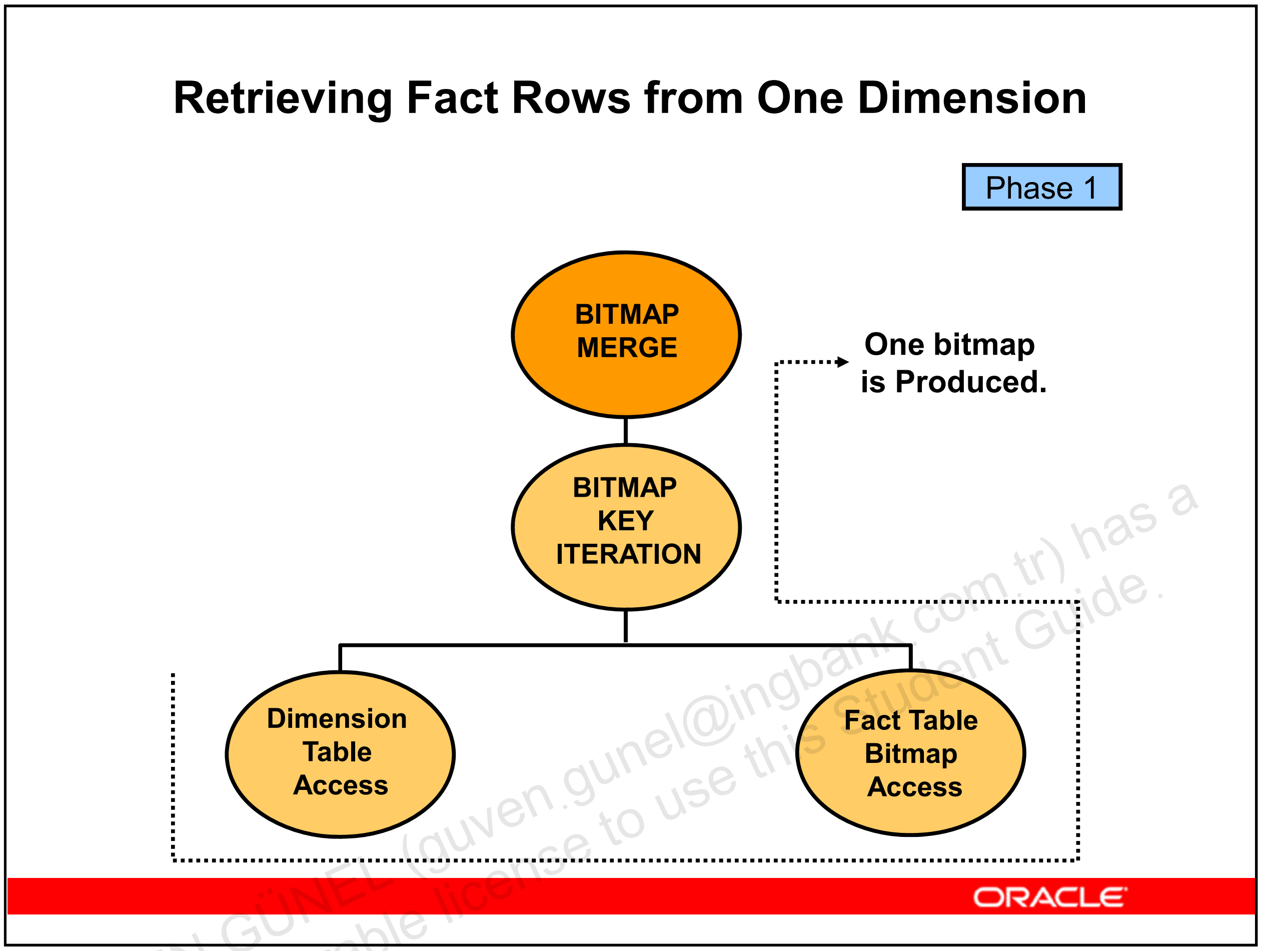
ORACLE

Star Transformation: Rewrite Example

The system processes the query seen earlier in two phases. In the first phase, the system uses the filters against the dimensional tables to retrieve the dimensional primary keys, which match those filters. Then the system uses those primary keys to probe the bitmap indexes on the foreign key columns of the fact table to identify and retrieve only the necessary rows from the fact table. That is, the system retrieves the result set from the sales table by using essentially the rewritten query in the slide.

Note: The SQL in the slide is a theoretical SQL statement that represents what goes on in phase I.

Retrieving Fact Rows from One Dimension



Retrieving Fact Rows from One Dimension

Retrieving Fact Rows from One Dimension

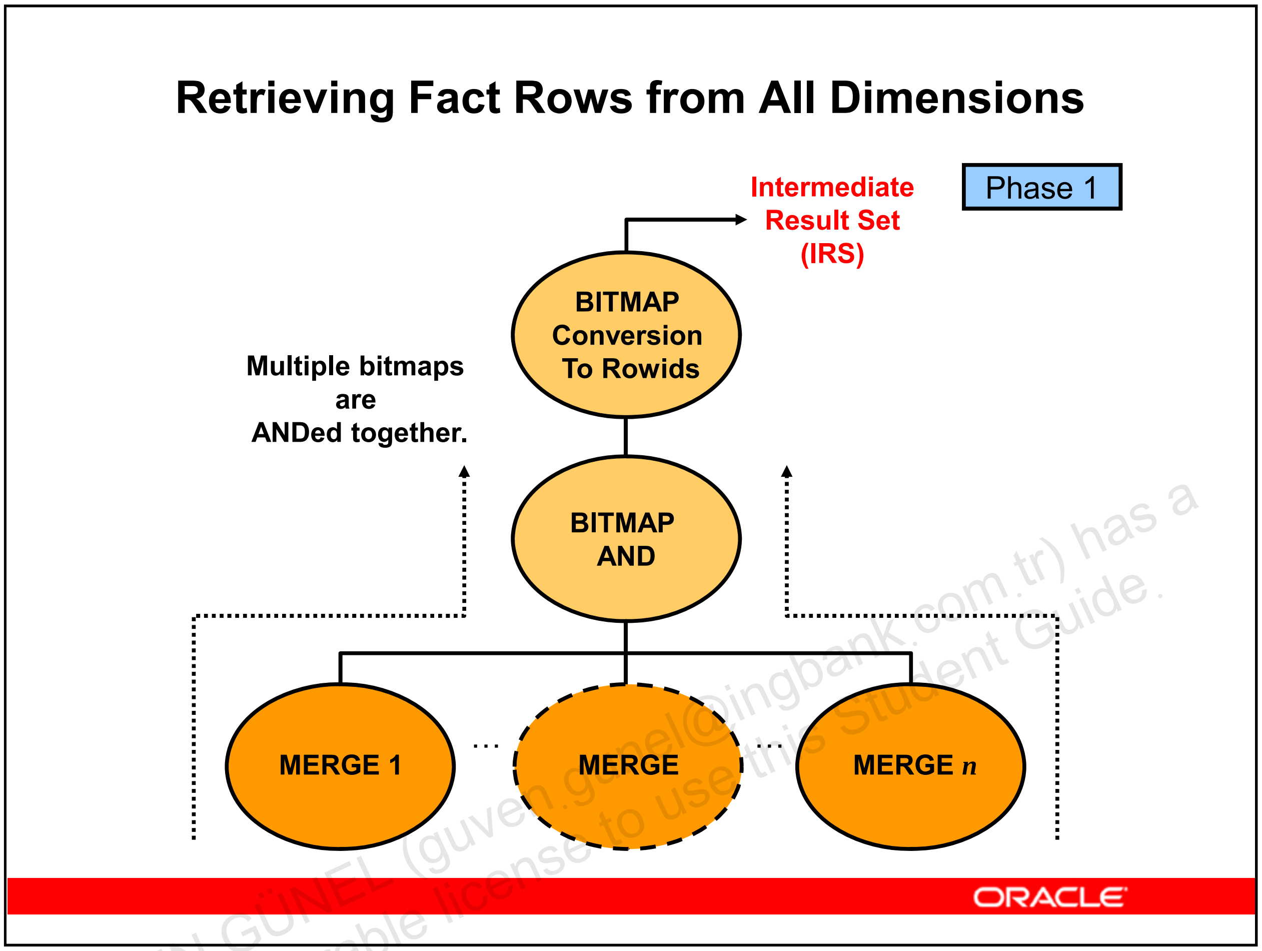
The slide shows retrieval of fact table rows using only one dimension table. Based on the corresponding dimension filter predicates, like in `t.calendar_quarter_desc IN ('1999-Q1','1999-Q2')` from the example in the previous slide, the system scans the dimension table, and for each corresponding row, it probes the corresponding fact bitmap index and fetches the corresponding bitmap.

`BITMAP KEY ITERATION` makes each key coming from its left input a lookup key for the index on its right input, and returns all bitmaps fetched by that index. Note that its left input supplies join keys from the dimension table in this case.

The last step in this tree merges all fetched bitmaps from the previous steps. This merge operation produces one bitmap that can be described as representing the rows of the fact table that join with the rows of interest from the dimension table.

Note: `BITMAP_MERGE_AREA_SIZE` plays an important role in tuning the performance of this operation when using the shared server mode. The system does not recommend using the `BITMAP_MERGE_AREA_SIZE` parameter unless the instance is configured with the shared server option. The system recommends that you enable automatic sizing of SQL working areas by setting `PGA_AGGREGATE_TARGET` instead. `BITMAP_MERGE_AREA_SIZE` is retained for backward compatibility.

Retrieving Fact Rows from All Dimensions

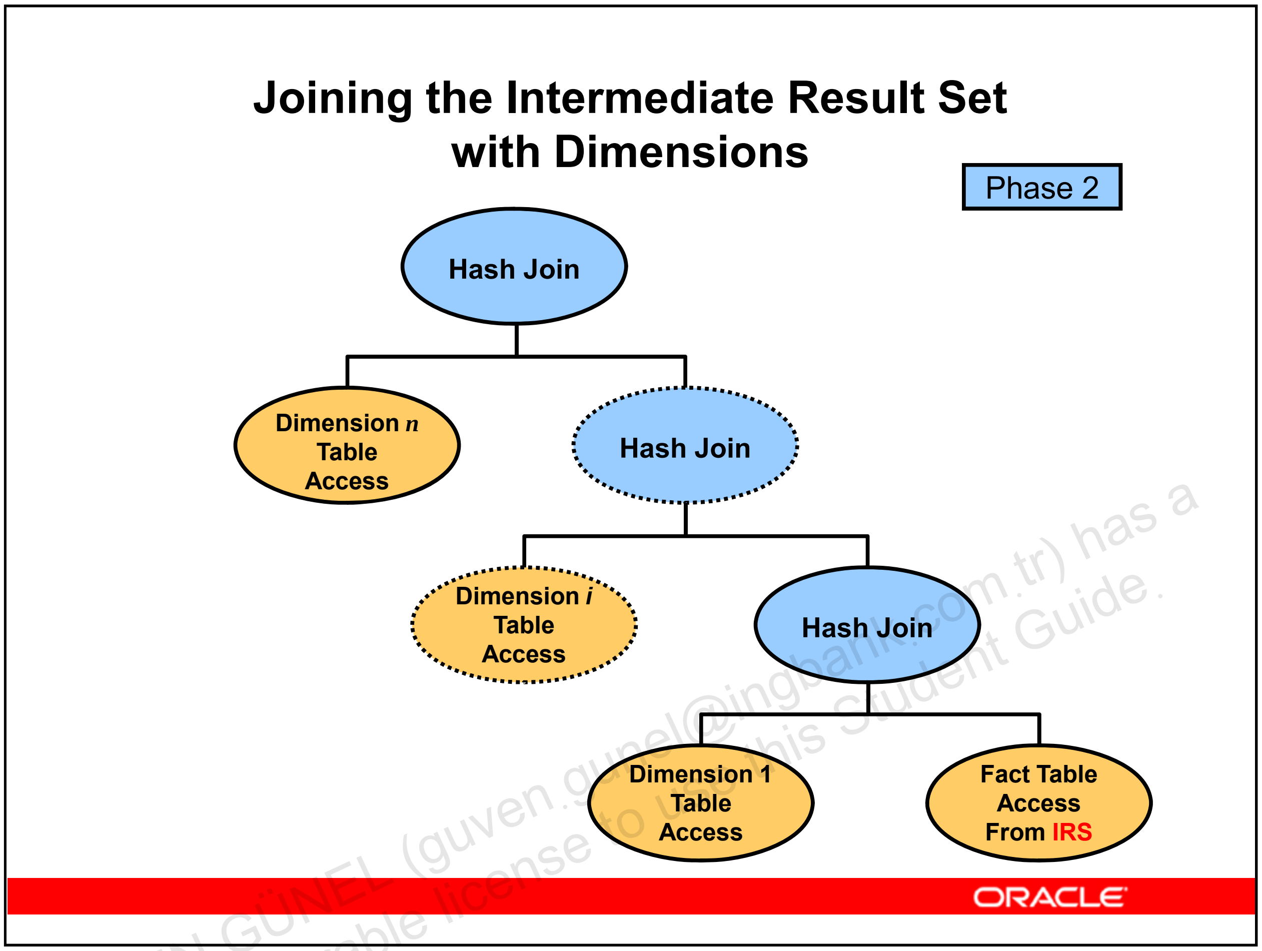

Retrieving Fact Rows from All Dimensions

Intermediate Result Set (IRS)
Phase 1
BITMAP Conversion To Rowids
Multiple bitmaps are ANDed together.
BITMAP AND
MERGE 1
...
MERGE
...
MERGE *n*
ORACLE

Retrieving Fact Rows from All Dimensions

During the first phase, the steps mentioned in the previous slide are repeated for each dimension table. So each `BITMAP MERGE` in the plan generates a bitmap for a single dimension table. To identify all rows from the fact table that are of interest, the system must intersect all generated bitmaps. This is to eliminate fact rows that join with one dimension, but not with all of them. This is achieved by performing a very efficient `BITMAP AND` operation on all the bitmaps generated for each dimension. The resulting bitmap can be described as representing the rows from the fact table that are known to join with all the qualified dimension rows.

Note: Until now, only fact bitmap indexes and dimension tables were used. To further access the fact table, the system must convert the generated bitmap to a rowids set.



Joining the Intermediate Result Set with Dimensions

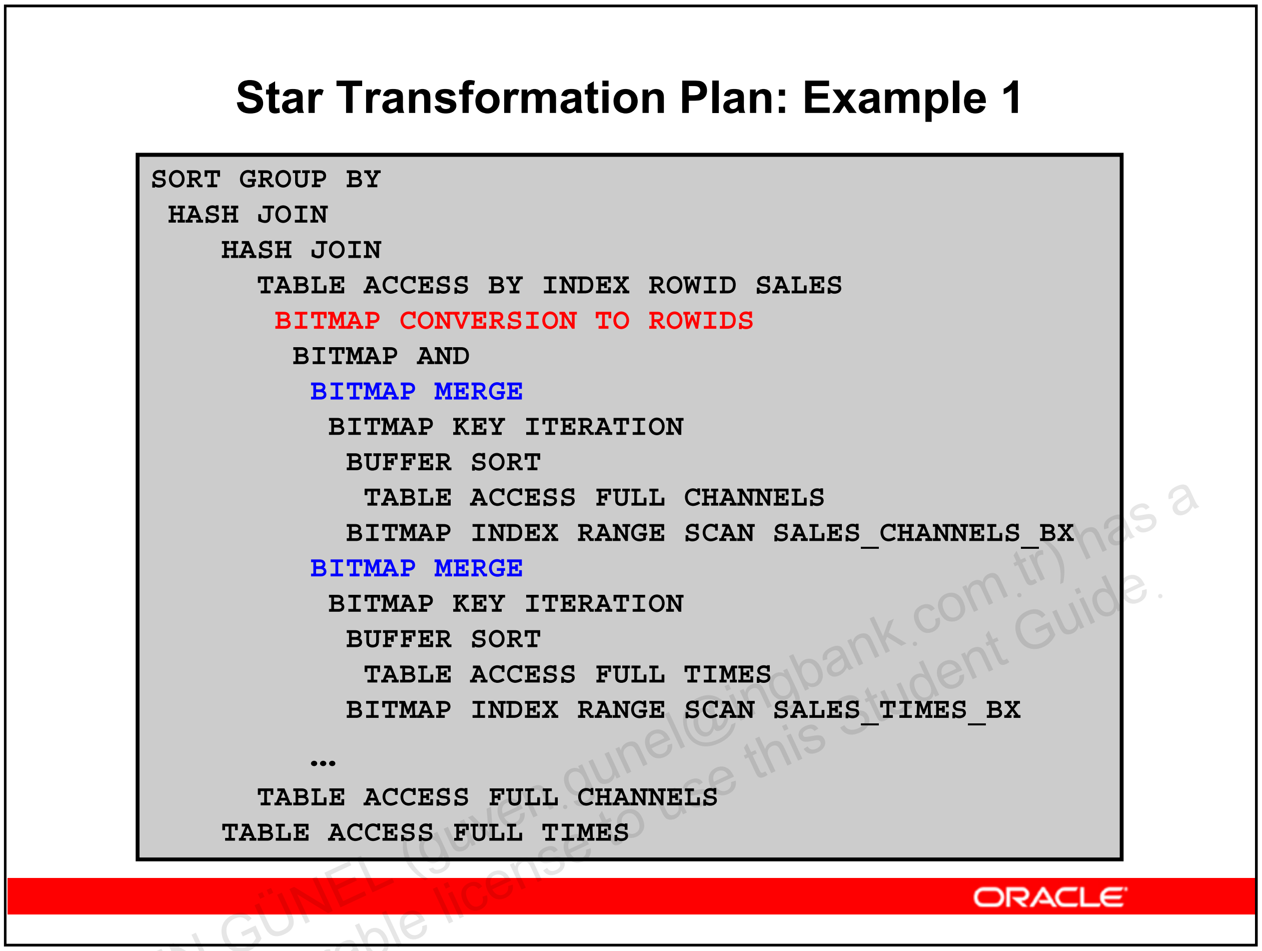
Phase 2
Hash Join
Dimension *n* Table Access
Hash Join
Dimension *i* Table Access
Hash Join
Dimension 1 Table Access
Fact Table Access From IRS
ORACLE

Joining the Intermediate Result Set with Dimensions

After the result set is determined, the system enters phase 2 of the star transformation algorithm. In this phase, it is needed to join the sales data, corresponding to the result set, with the dimension tables data used to group the rows and pertaining to the query's select list.

Note that the graphic in the slide shows that a hash join is performed between the fact table and its dimensions. Although a hash join is statistically the most-used technique to join rows in a star query, this might not be always true, as this is evaluated by the CBO.

Star Transformation Plan: Example 1



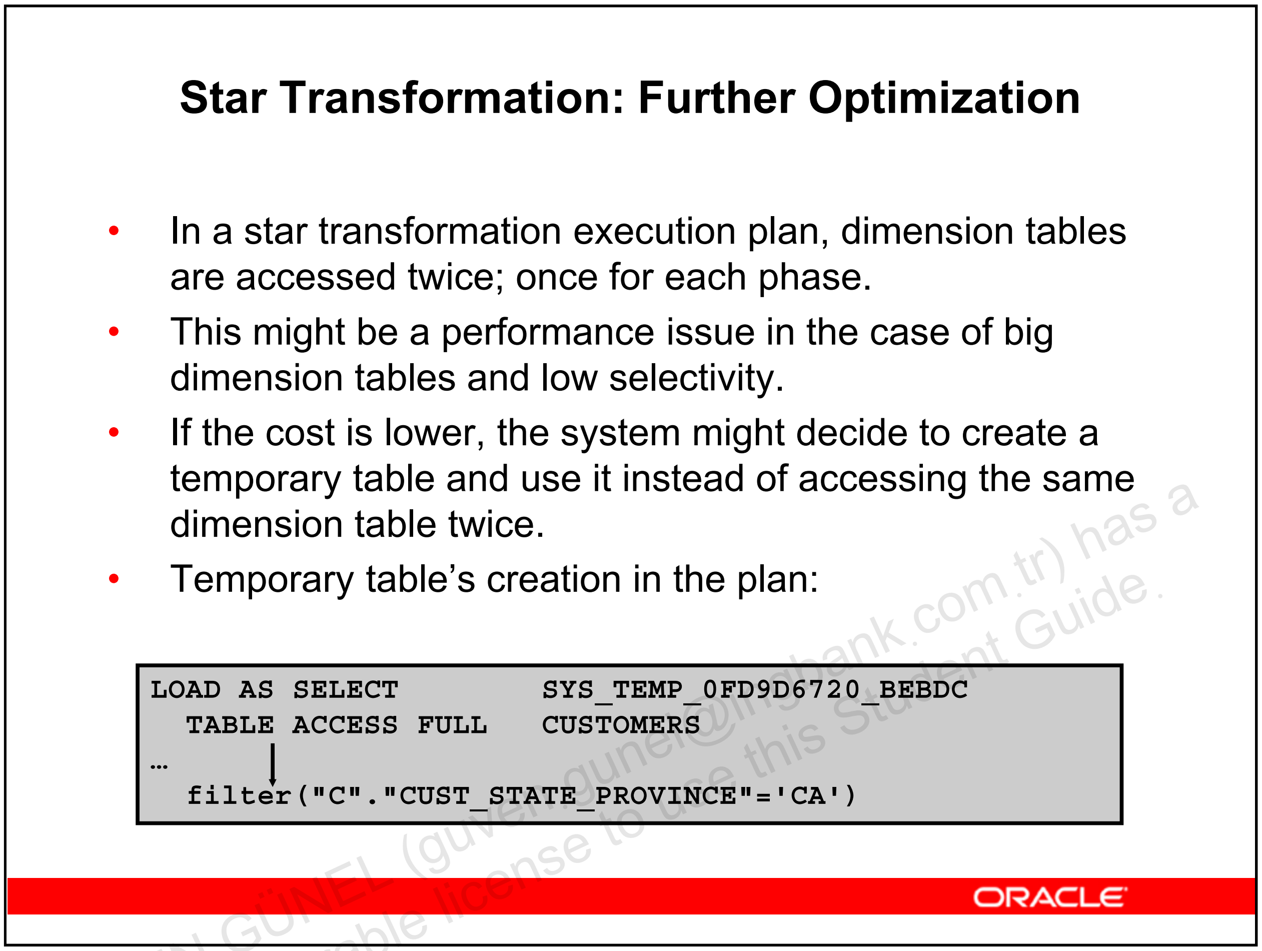
Star Transformation Plan: Example 1

```
SORT GROUP BY
 HASH JOIN
    HASH JOIN
      TABLE ACCESS BY INDEX ROWID SALES
       BITMAP CONVERSION TO ROWIDS
        BITMAP AND
         BITMAP MERGE
          BITMAP KEY ITERATION
           BUFFER SORT
            TABLE ACCESS FULL CHANNELS
           BITMAP INDEX RANGE SCAN SALES_CHANNELS_BX
         BITMAP MERGE
          BITMAP KEY ITERATION
           BUFFER SORT
            TABLE ACCESS FULL TIMES
           BITMAP INDEX RANGE SCAN SALES_TIMES_BX
         …
      TABLE ACCESS FULL CHANNELS
    TABLE ACCESS FULL TIMES
```

Star Transformation Plan: Example 1

This is a possible plan to answer the query shown in the “Execution Plan Without Star Transformation” section. Note that for formatting purposes, only the channels and times dimensions are shown. It is easy to generalize the case for *n* dimensions.

Note: It is supposed that `sales` is not partitioned.



Star Transformation: Further Optimization

- In a star transformation execution plan, dimension tables are accessed twice; once for each phase.
- This might be a performance issue in the case of big dimension tables and low selectivity.
- If the cost is lower, the system might decide to create a temporary table and use it instead of accessing the same dimension table twice.
- Temporary table's creation in the plan:

```
LOAD AS SELECT         SYS_TEMP_0FD9D6720_BEBDC
  TABLE ACCESS FULL    CUSTOMERS
…
  filter("C"."CUST_STATE_PROVINCE"='CA')
```

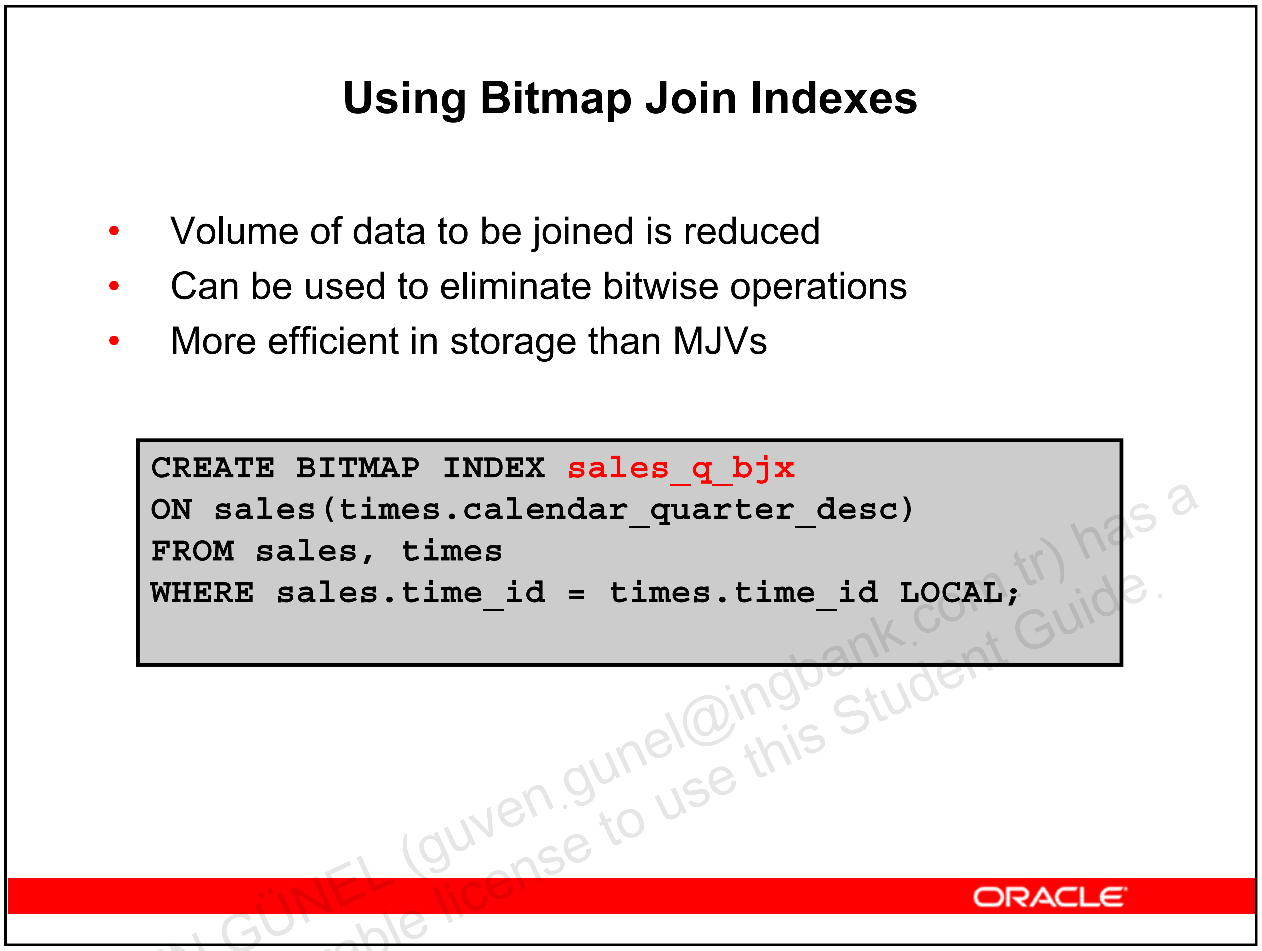
Star Transformation: Further Optimization

When you look at the previous execution plan, you see that each dimension table is accessed twice—once during the first phase, where the system determines the necessary fact table rows, and once when joining the fact rows to each dimension table during the second phase. This might be a performance issue if the dimension tables are big, and there is no fast access path to them for solving the problem. In such cases, the system might decide to create temporary tables containing information needed for both phases. This decision is made if the cost for creating a temporary table, consisting of the result set for both the predicate and the join columns on the dimension table, is cheaper than accessing the dimension table twice. In the previous execution plan example, the `TIMES` and `CHANNELS` tables are very small, and accessing them using a full table scan has a very small cost.

The creation of these temporary tables and the data insertion are shown in the execution plan. The name of those temporary tables is system-generated and varies. In the slide, you see an extract from an execution plan using temporary tables for the `CUSTOMERS` table.

Note: Temporary tables are not used by star transformation under the following conditions:

- The database is in read-only mode.
- The star query is part of a transaction that is in serializable mode.
- `STAR_TRANSFORMATION_ENABLED` is set to `TEMP_DISABLE`.



Using Bitmap Join Indexes

- Volume of data to be joined is reduced
- Can be used to eliminate bitwise operations
- More efficient in storage than MJVs

```
CREATE BITMAP INDEX sales_q_bjx
ON sales(times.calendar_quarter_desc)
FROM sales, times
WHERE sales.time_id = times.time_id LOCAL;
```

Using Bitmap Join Indexes

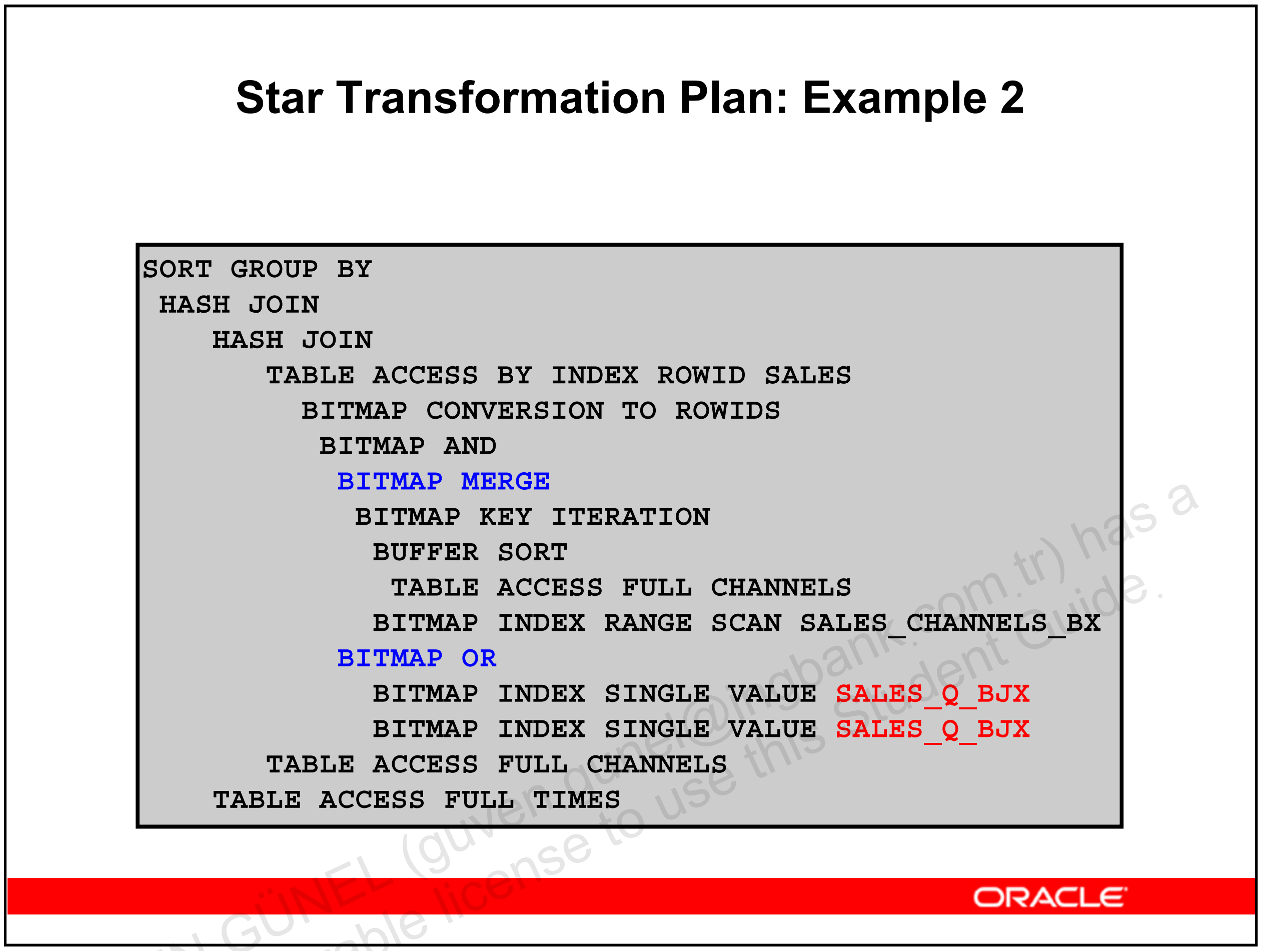
The volume of data that must be joined can be reduced if the join indexes used have already been precalculated.

In addition, the join indexes, which contain multiple dimension tables can eliminate bitwise operations, which are necessary in the star transformation with existing bitmap indexes.

Finally, bitmap join indexes are much more efficient in storage than materialized join views (MJVs), which do not compress rowids of the fact tables.

Assume that you have created the additional index structure mentioned in the slide.

Note: Since the SALES table is partitioned the bitmap join index will also be partitioned therefore the LOCAL keyword is required.



Star Transformation Plan: Example 2

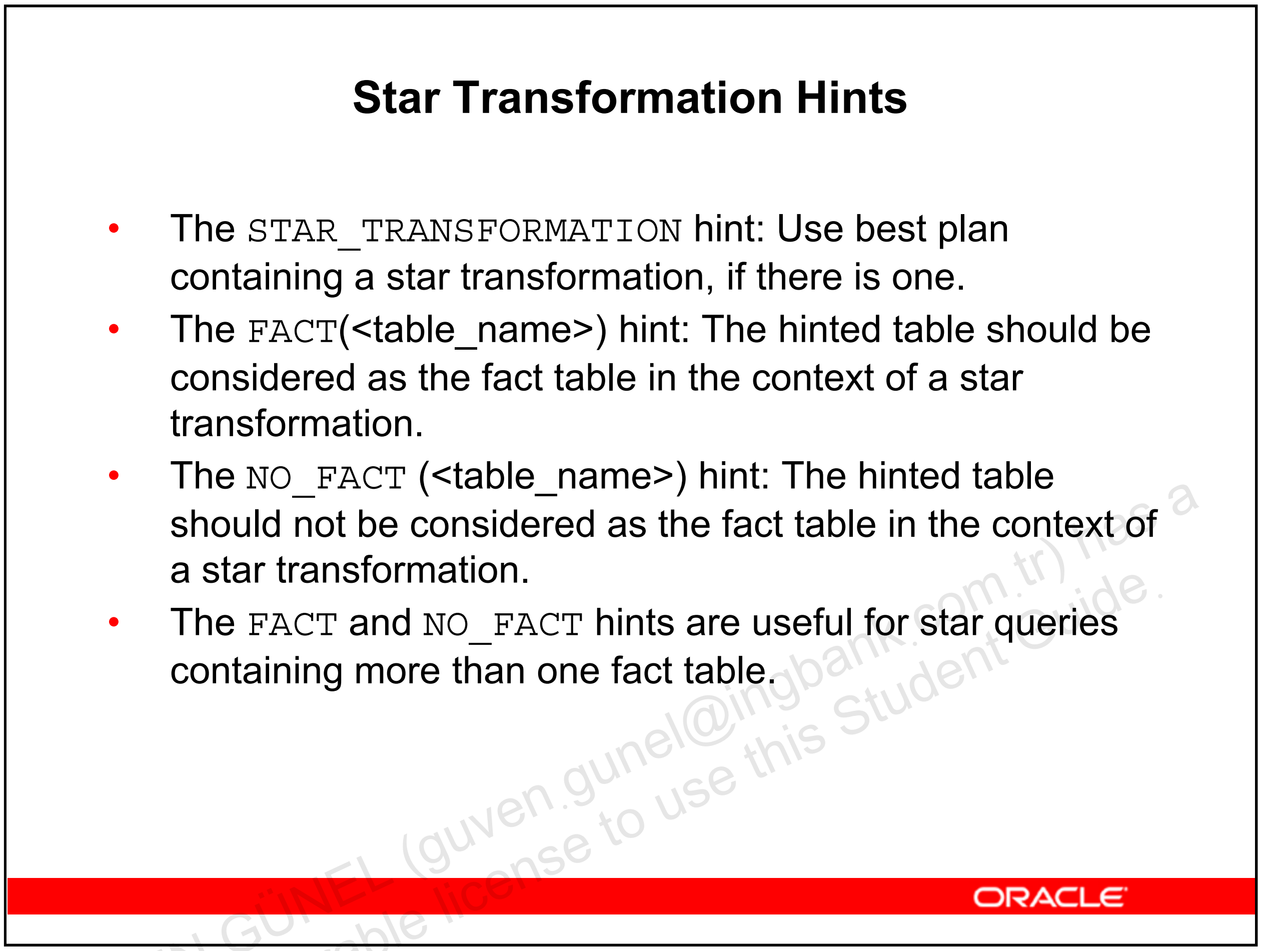
```
SORT GROUP BY
 HASH JOIN
    HASH JOIN
       TABLE ACCESS BY INDEX ROWID SALES
         BITMAP CONVERSION TO ROWIDS
          BITMAP AND
           BITMAP MERGE
            BITMAP KEY ITERATION
             BUFFER SORT
              TABLE ACCESS FULL CHANNELS
             BITMAP INDEX RANGE SCAN SALES_CHANNELS_BX
           BITMAP OR
             BITMAP INDEX SINGLE VALUE SALES_Q_BJX
             BITMAP INDEX SINGLE VALUE SALES_Q_BJX
       TABLE ACCESS FULL CHANNELS
    TABLE ACCESS FULL TIMES
```

Star Transformation Plan: Example 2

The processing of the same star query using the bitmap join index is similar to the previous example. The only difference is that the system uses the join index instead of a single-table bitmap index to access the `times` data in the first phase of the star query.

The difference between this plan as compared to the previous one is that the inner part of the bitmap index scan for the `times` dimension has no subselect in the rewritten query for phase 1. This is because the join predicate information on `times.calendar_quarter_desc` can be satisfied with the `sales_q_bjx` bitmap join index.

Note that access to the join index is done twice because the corresponding query's predicate is `t.calendar_quarter_desc IN ('1999-Q1','1999-Q2')`



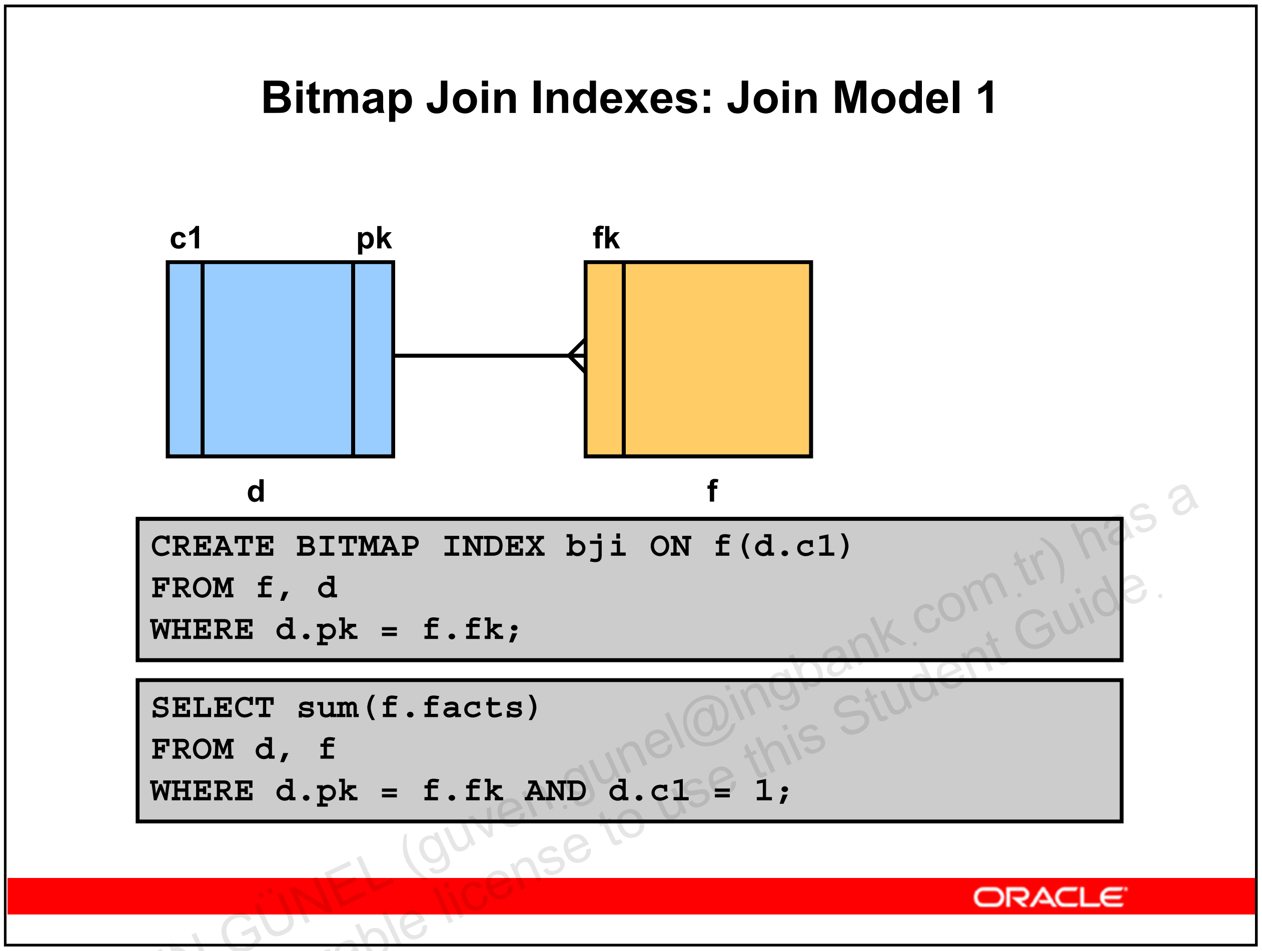
Star Transformation Hints

- The STAR_TRANSFORMATION hint: Use best plan containing a star transformation, if there is one.
- The FACT(<table_name>) hint: The hinted table should be considered as the fact table in the context of a star transformation.
- The NO_FACT (<table_name>) hint: The hinted table should not be considered as the fact table in the context of a star transformation.
- The FACT and NO_FACT hints are useful for star queries containing more than one fact table.

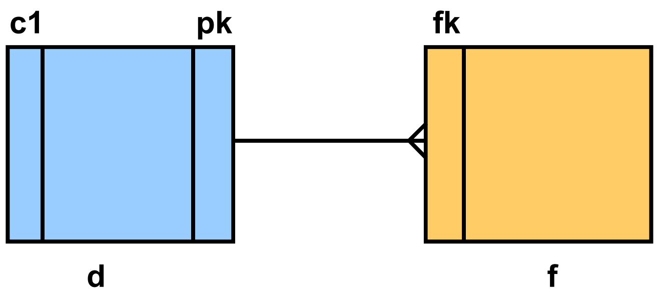
Star Transformation Hints

- The STAR_TRANSFORMATION hint makes the optimizer use the best plan in which the transformation has been used. Without the hint, the optimizer could make a cost-based decision to use the best plan generated without the transformation, instead of the best plan for the transformed query. Even if the hint is given, there is no guarantee that the transformation takes place. The optimizer only generates the subqueries if it seems reasonable to do so. If no subqueries are generated, there is no transformed query, and the best plan for the untransformed query is used, regardless of the hint.
- The FACT hint is used in the context of the star transformation to indicate to the transformation that the hinted table should be considered as a fact table and all other tables regardless of their size are considered as dimensions.
- The NO_FACT hint is used in the context of the star transformation to indicate to the transformation that the hinted table should not be considered as a fact table.

Note: The FACT and NO_FACT hints might be useful only in case there are more than one fact table accessed in the star query.



Bitmap Join Indexes: Join Model 1



```
CREATE BITMAP INDEX bji ON f(d.c1)
FROM f, d
WHERE d.pk = f.fk;
```

```
SELECT sum(f.facts)
FROM d, f
WHERE d.pk = f.fk AND d.c1 = 1;
```

ORACLE

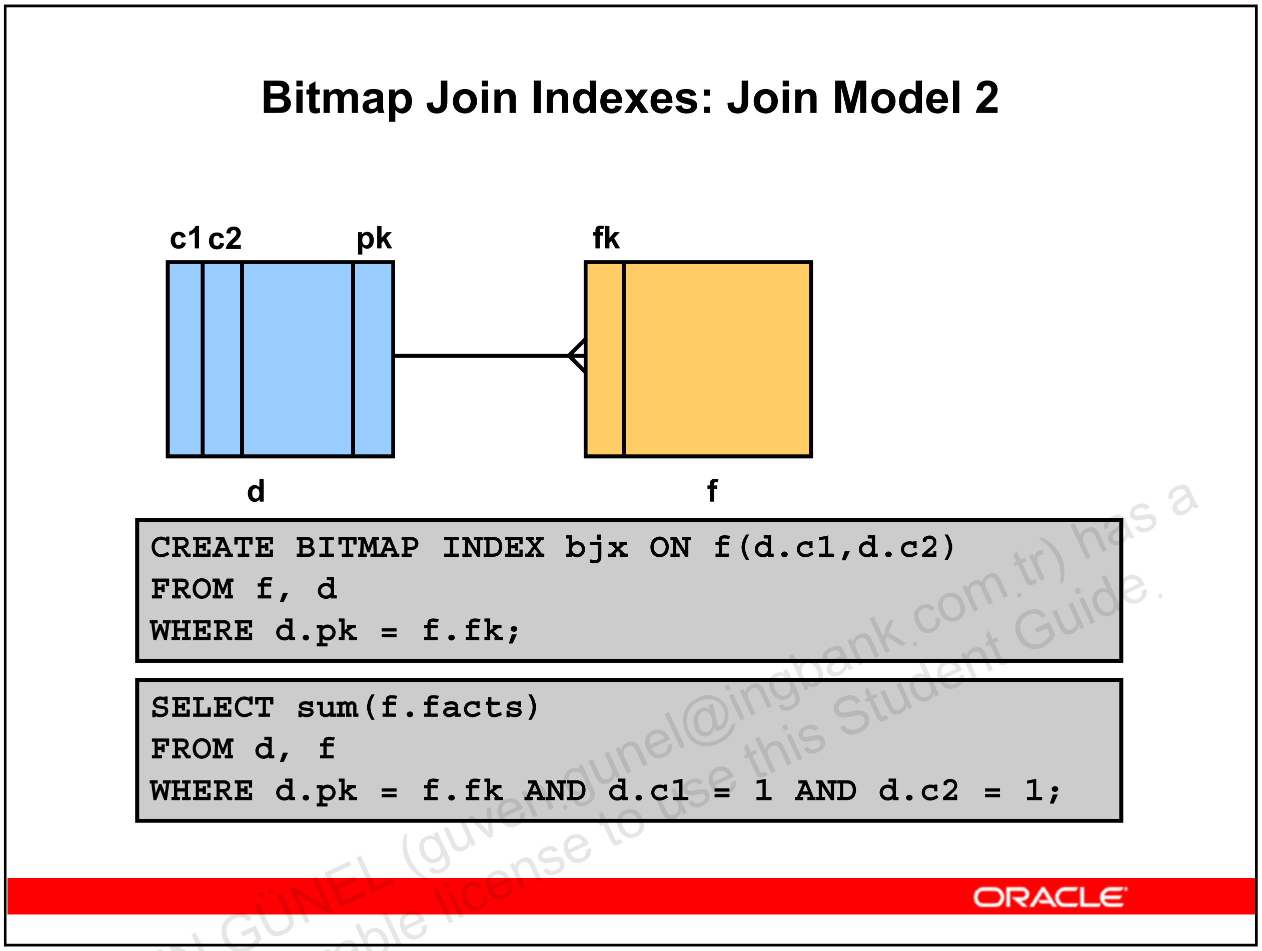
Bitmap Join Indexes: Join Model 1

In the following three slides, `F` represents the fact table, `D`, the dimension table, `PK` a primary key, and `FK` a foreign key.

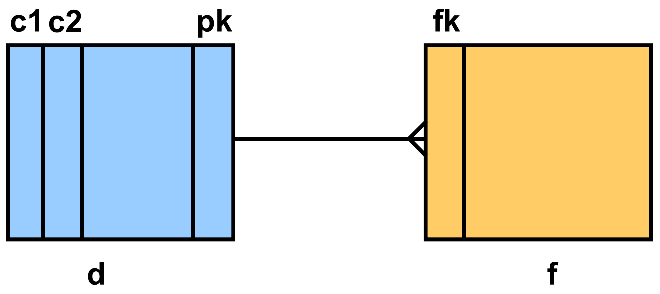
A bitmap join index can be used in the `SELECT` statement in the slide to avoid the join operation.

Similar to the materialized join view, a bitmap join index precomputes the join and stores it as a database object. The difference is that a materialized join view materializes the join into a table while a bitmap join index materializes the join into a bitmap index.

Note: `C1` is the indexed column in the dimension table.



Bitmap Join Indexes: Join Model 2



```
CREATE BITMAP INDEX bjx ON f(d.c1,d.c2)
FROM f, d
WHERE d.pk = f.fk;
```

```
SELECT sum(f.facts)
FROM d, f
WHERE d.pk = f.fk AND d.c1 = 1 AND d.c2 = 1;
```

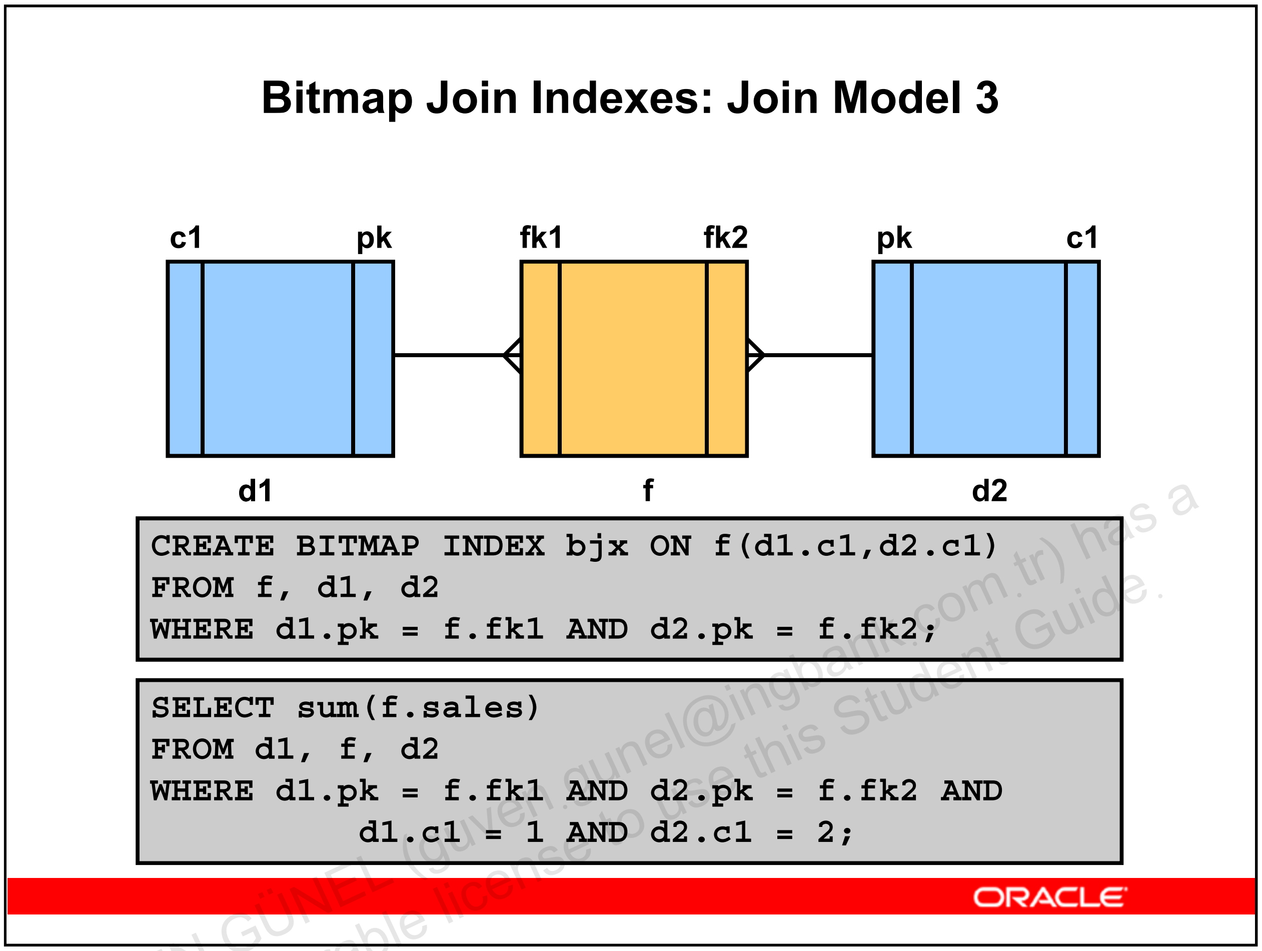
Bitmap Join Indexes: Join Model 2

The model in the slide is an extension of model 1, requiring a concatenated bitmap join index to represent it.

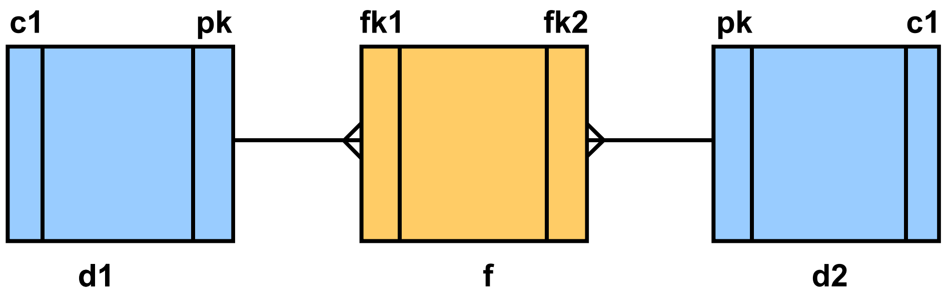
Note that BJX, in this case, can also be used to answer the following select statement:

```
select sum(f.facts) from d,f where d.pk=f.fk and d.c1=1
```

This is due to the fact that D.C1 is the leading part of the BJX.



Bitmap Join Indexes: Join Model 3



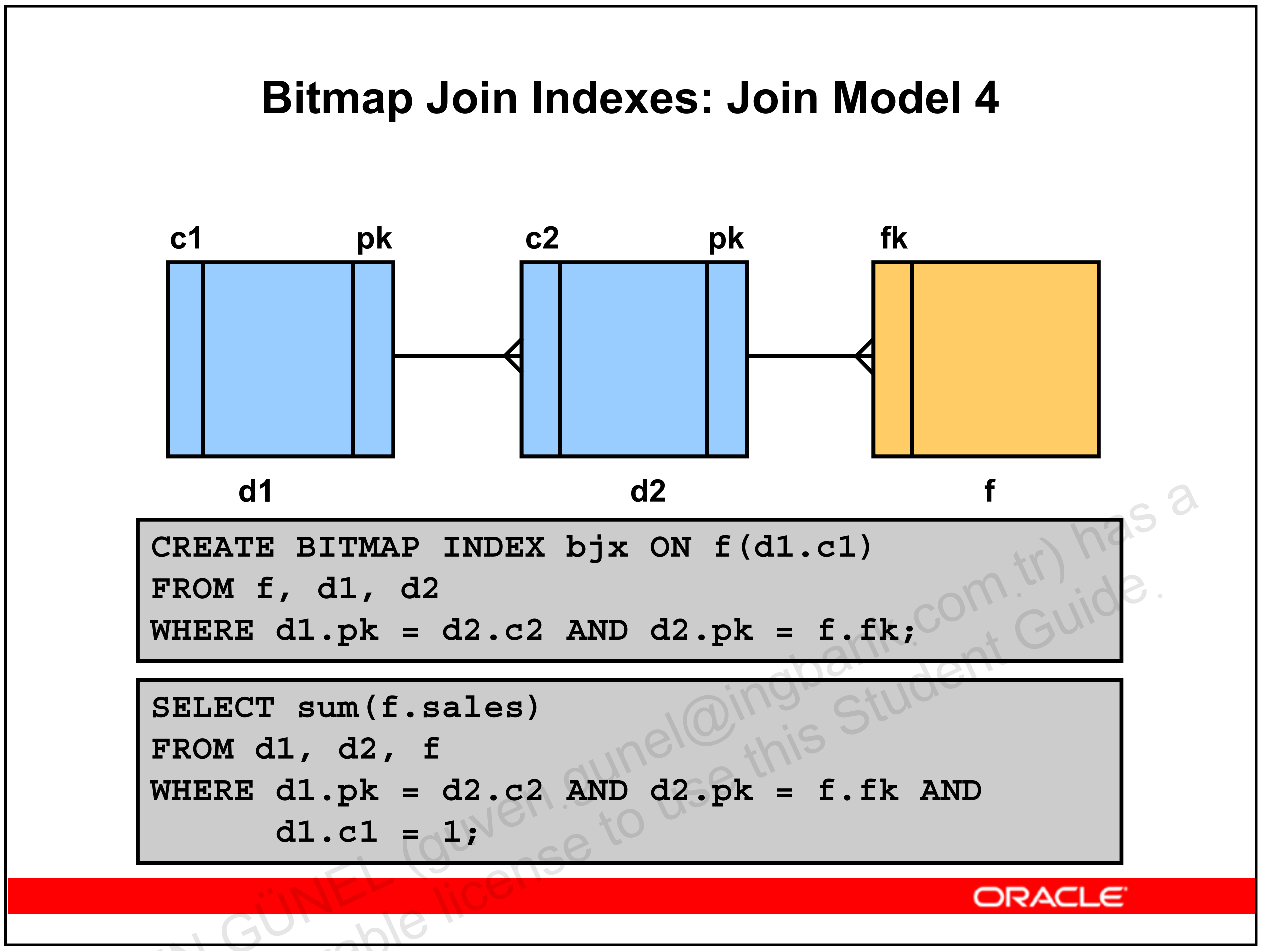
```
CREATE BITMAP INDEX bjx ON f(d1.c1,d2.c1)
FROM f, d1, d2
WHERE d1.pk = f.fk1 AND d2.pk = f.fk2;
```

```
SELECT sum(f.sales)
FROM d1, f, d2
WHERE d1.pk = f.fk1 AND d2.pk = f.fk2 AND
          d1.c1 = 1 AND d2.c1 = 2;
```

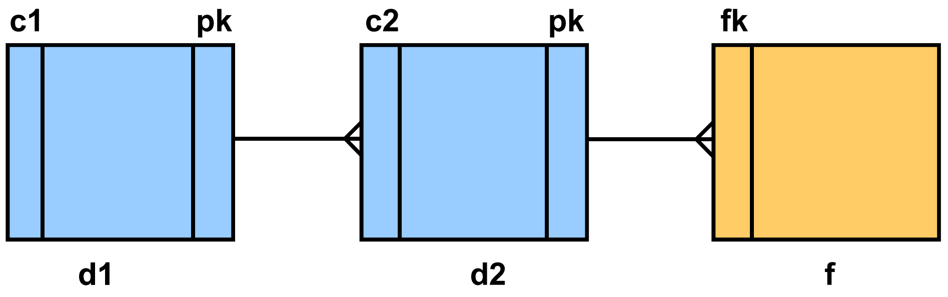
Bitmap Join Indexes: Join Model 3

This model also requires the concatenated bitmap join index shown in the slide. In this case, two dimension tables are used.

Bitmap Join Indexes: Join Model 4



Bitmap Join Indexes: Join Model 4



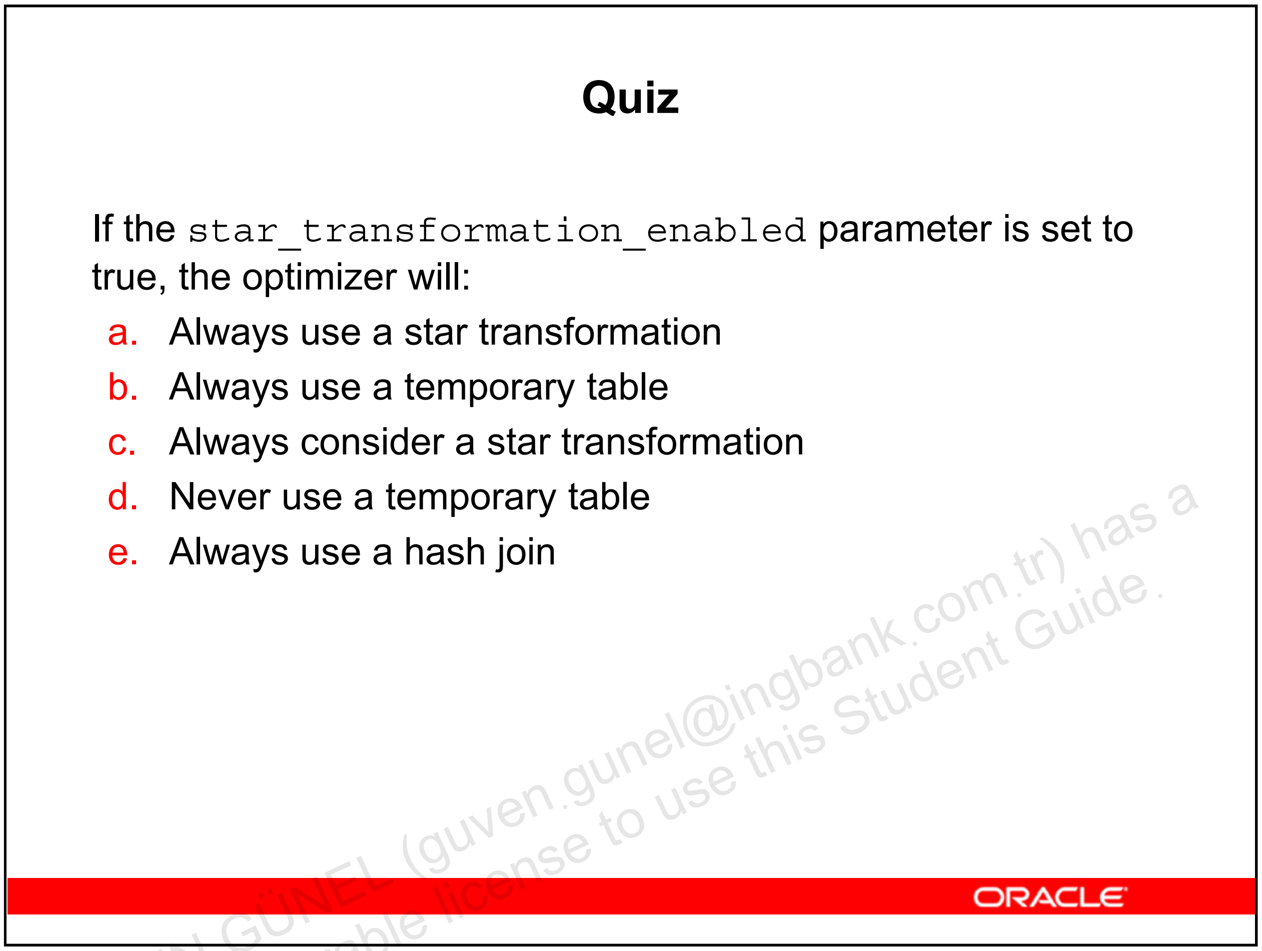
```
CREATE BITMAP INDEX bjx ON f(d1.c1)
FROM f, d1, d2
WHERE d1.pk = d2.c2 AND d2.pk = f.fk;
```

```
SELECT sum(f.sales)
FROM d1, d2, f
WHERE d1.pk = d2.c2 AND d2.pk = f.fk AND
      d1.c1 = 1;
```

Bitmap Join Indexes: Join Model 4

The slide shows a snowflake model that involves joins between two or more dimension tables. It can be expressed by a bitmap join index. The bitmap join index can be either single or concatenated depending on the number of columns in the dimension tables to be indexed. A bitmap join index on `D1.C1` with a join between `D1` and `D2` and a join between `D2` and `F` can be created as shown in the slide with `BJX`.

Quiz

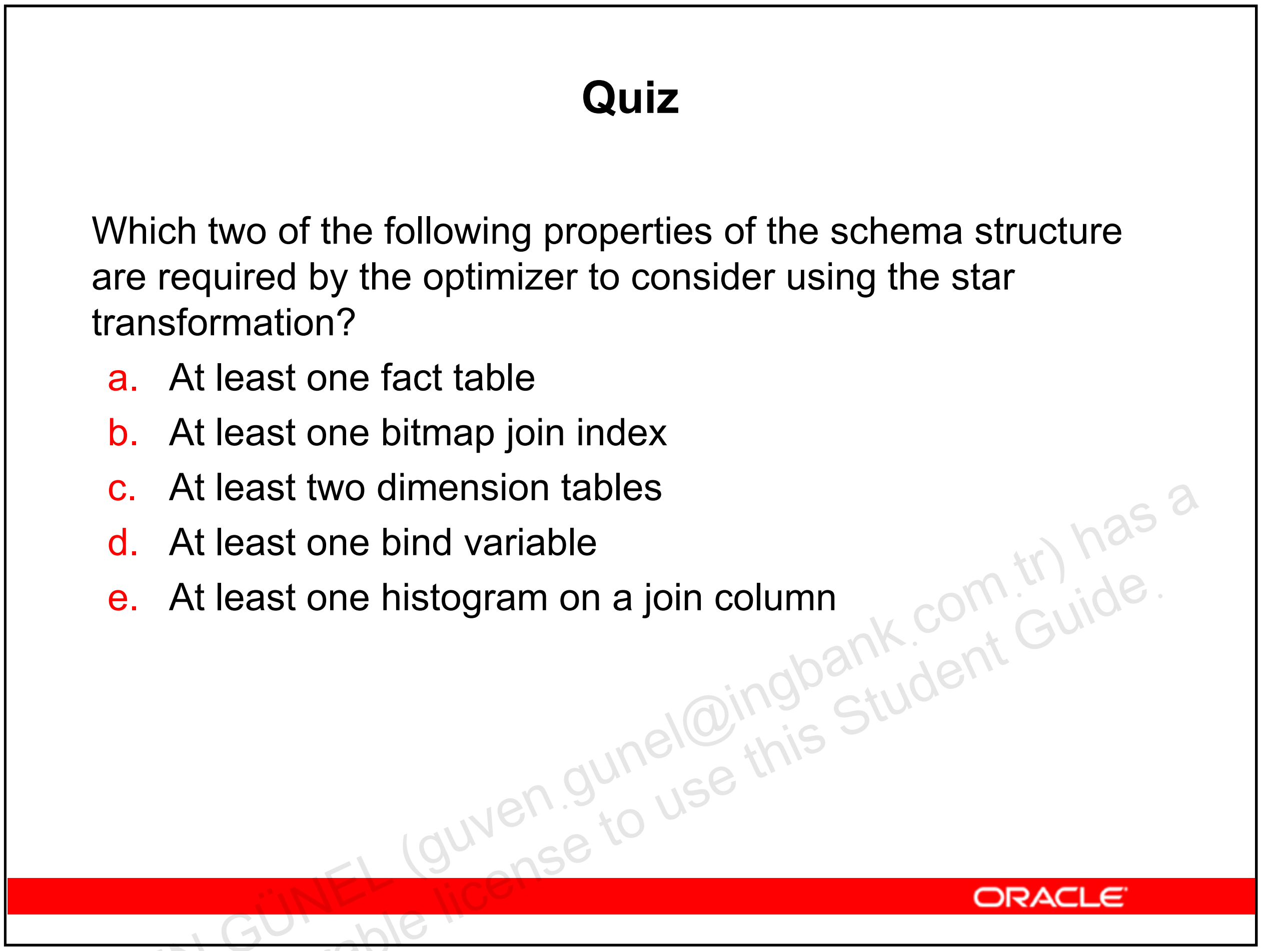


Quiz

If the `star_transformation_enabled` parameter is set to true, the optimizer will:

a. Always use a star transformation
b. Always use a temporary table
c. Always consider a star transformation
d. Never use a temporary table
e. Always use a hash join

Answer: c



Quiz

Which two of the following properties of the schema structure are required by the optimizer to consider using the star transformation?

a. At least one fact table
b. At least one bitmap join index
c. At least two dimension tables
d. At least one bind variable
e. At least one histogram on a join column

Answer: a, c

Quiz

Assuming that the `START_TRANSFORMATION_ENABLED` parameter is set to `TRUE`, the star transformation is chosen by the optimizer when:

a. All the conditions are met

b. The cost is lower than a tradition access path

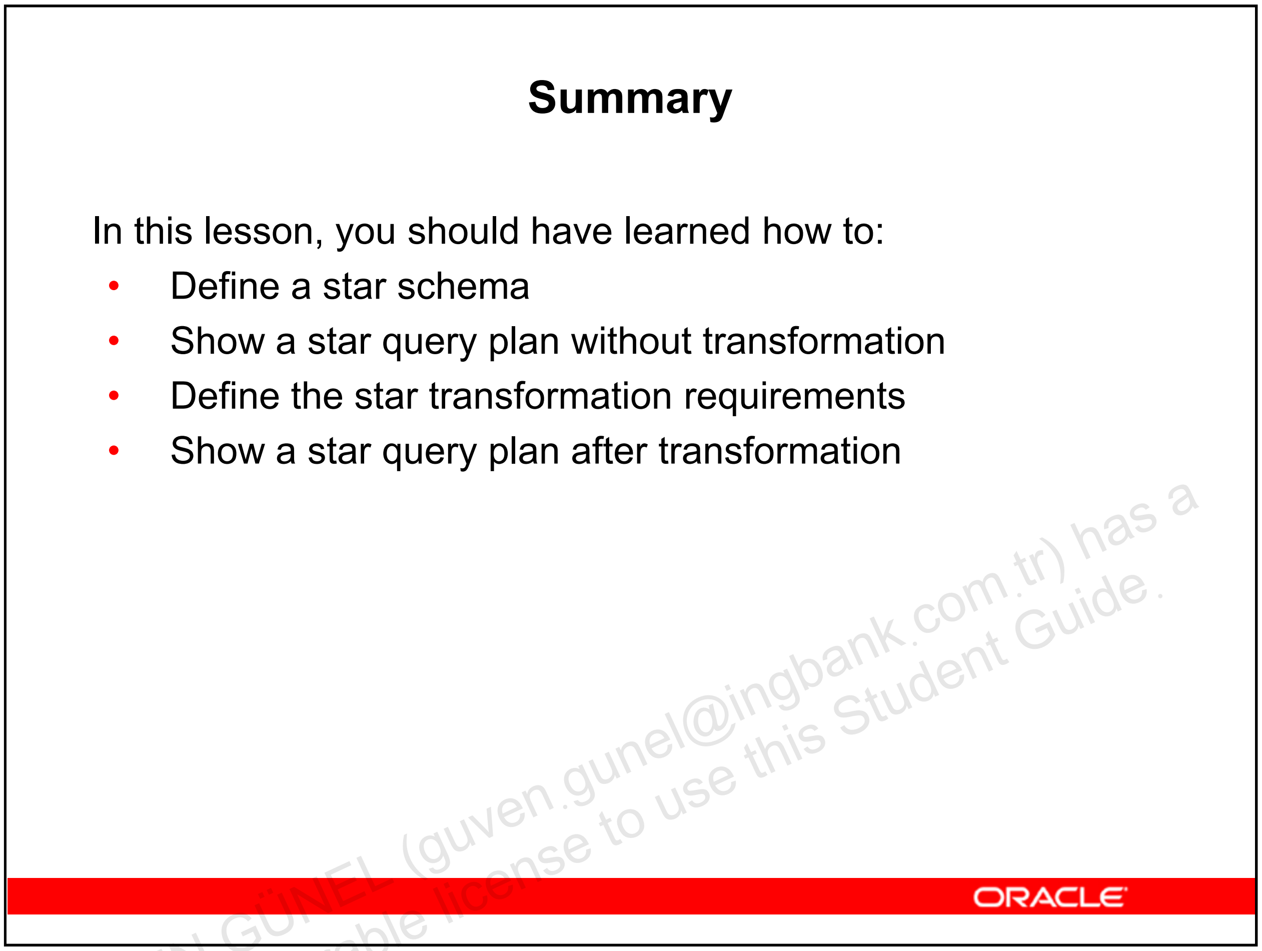
c. The bitmap join indexes eliminate additional joins

d. Temporary tables can be used to reduce table accesses

ORACLE

Answer: b

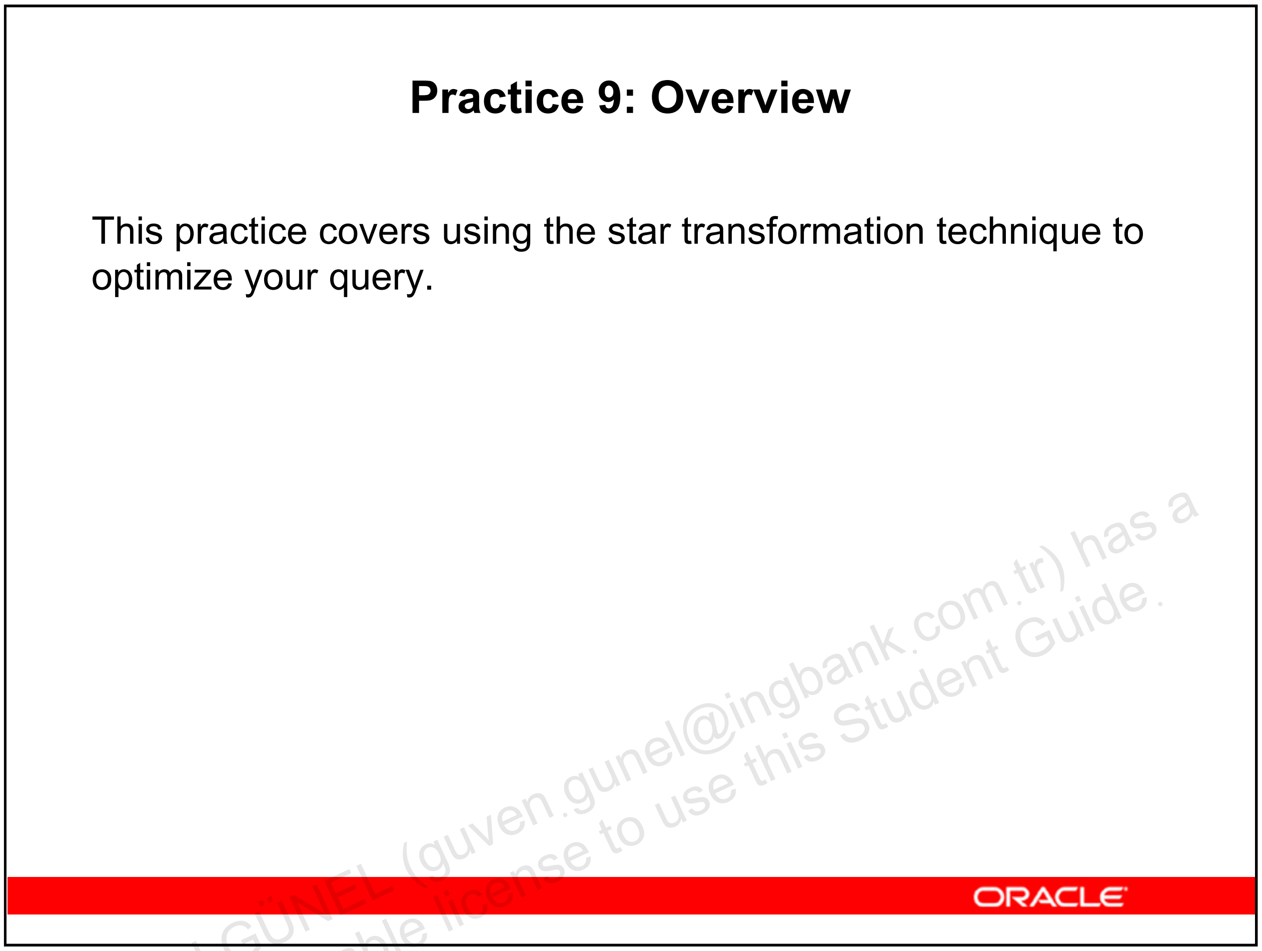
Summary



Summary

In this lesson, you should have learned how to:
- Define a star schema
- Show a star query plan without transformation
- Define the star transformation requirements
- Show a star query plan after transformation

Practice 9: Overview



Practice 9: Overview

This practice covers using the star transformation technique to optimize your query.