

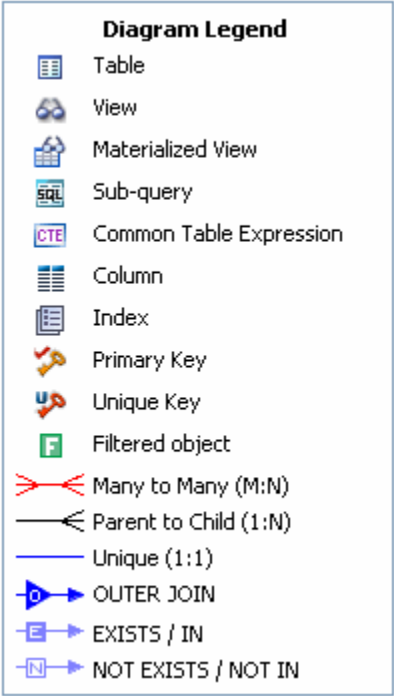
# Interpreting the VST diagram graphics

This section contains the following topics that will help you understand the graphics in VST diagrams:

- [Viewing the Diagram Legend](#)
- [Colors](#)
- [Connecting lines/joins](#)
- [Viewing object SQL](#)
- [Refreshing tuning statements](#)
- [Refreshing the VST diagram](#)

## Viewing the Diagram Legend

Click the **Diagram Legend Toggle** button as shown in the diagram below to see a description of the icons and relationship lines used in VST diagrams.



## Colors

The color of the index entries in the **Collect and Create Indexes** table is interpreted as follows:

| Text Color | Interpretation                                                                |
|------------|-------------------------------------------------------------------------------|
| Green      | Index is used in the query.                                                   |
| Blue       | Index is usable but not used by the current execution path.                   |
| Orange     | This index is missing. SQL Query Tuner recommends that you create this index. |
| Grey       | This index exists on the table but not usable in this query as it is written. |

## Connecting lines/joins

Joins are represented with connecting lines between nodes. You can move tables in the diagram by clicking and dragging them to the desired location. The position of the connecting lines is automatically adjusted. The following describes when a particular type of connecting line is used and the default positioning of the line.

| Connecting Lines | When Used |
|------------------|-----------|
|------------------|-----------|

|  |                                                                                                                                                                                                                                                                                           |
|--|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <b>One-to-One</b> join relationships are graphed horizontally using blue lines. For more information, see <a href="#">One-to-One Join</a> .                                                                                                                                               |
|  | <b>One-to-Many</b> join relationships are graphed with the many table above the one table. For more information, see <a href="#">One-to-Many Join</a> .                                                                                                                                   |
|  | <b>Cartesian Join</b> shows the table highlighted in red with no connectors to indicate that it is joined in via a Cartesian join. For more information, see <a href="#">Cartesian Join</a> .                                                                                             |
|  | <b>Many-to-Many Join</b> relationships are connected by a red line and the relative location is not restricted. For more information, see <a href="#">Many-to-Many Join</a> .                                                                                                             |
|  | <b>Indirect Relationship</b> . For more information, see <a href="#">Indirect Relationship</a> .                                                                                                                                                                                          |
|  | <b>Outer Join</b> . For more information, see <a href="#">Outer Join</a> .                                                                                                                                                                                                                |
|  | <b>Unique</b> . For more information, see <a href="#">Unique</a> .                                                                                                                                                                                                                        |
|  | <b>Not Exists and Not in</b> relationship lines connect the subquery to the table being queried.<br><br>Notice that when you click this relationship line, the SQL text creating the relationship is also selected. For more information, see <a href="#">Not In or Not Exists Join</a> . |
|  | <b>Exists and In</b> relationship lines connect the subquery to the table being queried.<br><br>Notice that when you click this relationship line, the SQL text creating the relationship is also selected. For more information, see <a href="#">In or Exists Join</a> .                 |

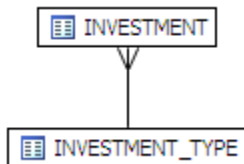
## One-to-one join

If two tables are joined on their primary key, then graphically, these would be laid out side-by-side, with a one-to-one connector:



## One-to-many join

This is the default positioning of a one-to-many relationship, where INVESTMENT\_TYPE is the master table and INVESTMENT is the details table.



The following is an example of a query that consists of only many-to-one joins, which is more typical:

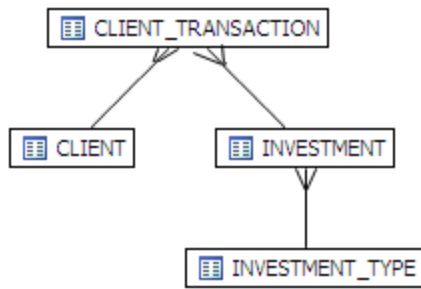
```

SELECT
    ct.action, c.client_id,
    i.investment_unit,
    it.investment_type_name

FROM
    client_transaction ct,
    client c,
    investment_type it,
    investment i

WHERE
    ct.client_id = c.client_id AND
    ct.investment_id = i.investment_id AND
    i.investment_type_id = it.investment_type_id and
    client_transaction_id=1

```



## Cartesian join

A Cartesian join is described in the following example where the query is missing join criteria on the table INVESTMENT:

```

SELECT

    A.BROKER_ID BROKER_ID, A.BROKER_LAST_NAME
    BROKER_LAST_NAME, A.BROKER_FIRST_NAME
    BROKER_FIRST_NAME, A.YEARS_WITH_FIRM
    YEARS_WITH_FIRM, C.OFFICE_NAME OFFICE_NAME,
    SUM (B.BROKER_COMMISSION) TOTAL_COMMISSIONS

FROM

    BROKER A,
    CLIENT_TRANSACTION B,
    OFFICE_LOCATION C,
    INVESTMENT I

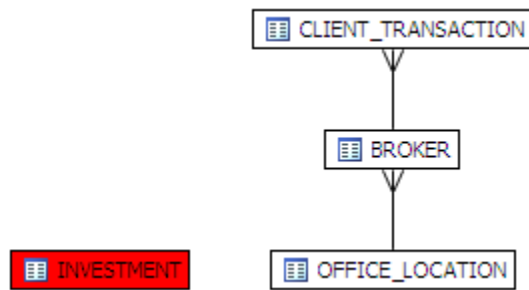
WHERE

    A.BROKER_ID = B.BROKER_ID AND
    A.OFFICE_LOCATION_ID = C.OFFICE_LOCATION_ID

GROUP BY

    A.BROKER_ID,
    A.BROKER_LAST_NAME,
    A.BROKER_FIRST_NAME,
    A.YEARS_WITH_FIRM,
    C.OFFICE_NAME;
  
```

Graphically, this looks like:



INVESTMENT is highlighted in red with no connectors to indicate that it is joined in via a Cartesian join.

Possible missing join conditions are displayed in the **Overview** tab under **Generated Cases** in the transformations area. DB Optimize recommends that you create these joins.

| SQL Statements and Cases                       |                     | Cost    | Execution Statistics | Other Execution Statistics |               |       |
|------------------------------------------------|---------------------|---------|----------------------|----------------------------|---------------|-------|
| Name                                           | Text                | Value   | Elapsed Time (s)     | Physical Reads             | Logical Reads | CPU T |
| SQL SELECT 1                                   | select from BROKER, | 34014.0 | 6.22                 | 0                          | 170           |       |
| [Missing a valid join criteria] transformation |                     | 274.0   | 0.04                 | 0                          | 167           |       |
| ----- FULL                                     |                     | 34019.0 | 6.29                 | 0                          | 173           |       |
| ----- LEADING1                                 |                     | 34017.0 | 6.25                 | 0                          | 192           |       |
| ----- ALL_ROWS                                 |                     | 34014.0 | 6.35                 | 0                          | 170           |       |
| ----- LEADING3                                 |                     | 34017.0 | 6.41                 | 0                          | 170           |       |
| ----- INDEX                                    |                     | 34392.0 | 6.58                 | 0                          | 414           |       |
| ----- LEADING2                                 |                     | 38148.0 | 7.94                 | 0                          | 170           |       |
| ----- ORDERED                                  |                     | 38147.0 | 8.61                 | 0                          | 170           |       |
| ----- USE_NL                                   |                     | 38198.0 | 9.03                 | 0                          | 37518         |       |

**i** Transformations are highlighted in yellow.

## Implied Cartesian join

If there are different details for a master without other criteria then a Cartesian-type join is created:

```
SELECT *
FROM
    investment i,
    broker b,
    client c
WHERE
    b.manager_id=c.client_id and
    i.investment_type_id=c.client_id;
```



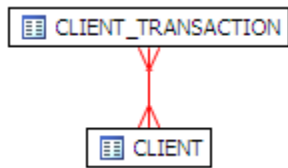
The result set of BROKER to CLIENT will be multiplied by the result set of INVESTMENT to CLIENT.

## Many-to-many join

If there is no unique index at either end of a join then it can be assumed that in some or all cases the join is many-to-many; there are no constraints preventing a many-to-many join. For example, examine the following query:

```
SELECT *
FROM
    client_transaction ct, client c
WHERE
    ct.transaction_status=c.client_marital_status;
```

There is no unique index on either of the fields being joined so the optimizer assumes this is a many-to-many join and the relationship is displayed graphically as:



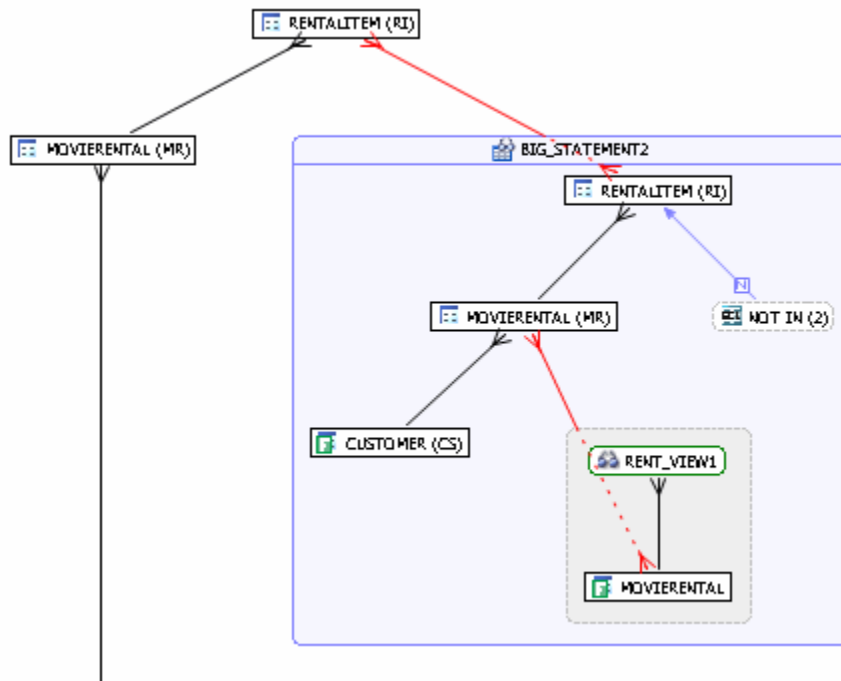
If one of the fields is unique, then the index should be declared as such to help the optimizer.

## Indirect relationship

Indirect relationships are produced by the following SQL, where BIG\_STATEMENT2 is a Materialized View.

```
SELECT CS.*
FROM
    MOVIES.CUSTOMER CS,
    MOVIES.MOVIERENTAL MR,
    MOVIES.RENTALITEM RI,
    OE.BIG_STATEMENT2
WHERE
    CS.ZIP > '75062' AND MR.RENTALID = RI.RENTALID
    AND RI.ITEMNUMBER = OE.BIG_STATEMENT2.ITEMNUMBER
    AND MR.CUSTOMERID = CS.CUSTOMERID;
```

The following diagram produced by the SQL above shows that an indirect relationship exists between the RENTALITEM(RI) tables inside and outside the materialized view, BIG\_STATEMENT2. An indirect relationship also exists between MOVIERENTAL (MR) inside BIG\_STATEMENT2 and MOVIERENTAL (MR) inside the RENT\_VIEW1 view.



## In or exists join

The following SQL contains a nest IN subquery (shown in bold text) that is graphically represented with the Subquery summary icon and the IN join.

```
SELECT
    cs.customerid, cs.firstname, cs.lastname,
    mr.rentalid, mr.duedate, mr.totalcharge, ri.itemnumber
FROM
    (
        SELECT
```

```

        c1.customerid, c1.firstname, c1.lastname, c1.phone
FROM
    MOVIES.customer c1
WHERE
    EXISTS (SELECT NULL) cs, (FROM MOVIES.customer c2
WHERE
    c1.customerid <>c2.customerid AND c1.lastname=c2.lastname AND c1.phone BETWEEN 0 AND 9999569900)
SELECT
    customerid, rentalid, duedate, totalcharge, rentaldate
FROM
    MOVIES.movierental
WHERE
    totalcharge > 10
    )
    mr, MOVIES.rentalitem ri
WHERE
    LENGTH (cs.lastname) = 10
    AND 1 < cs.customerid
    AND ROUND (ri.rentalid) > 10
    AND TRUNC (ri.itemnumber) > 1
    AND mr.totalcharge > (SELECT AVG (totalcharge)
FROM
    MOVIES.movierental
WHERE
    TOTALCHARGE >= 40)
    AND ri.moviecopyid
    NOT IN (SELECT mc.moviecopyid FROM MOVIES.moviecopy mc
WHERE
    mc.copyformat = 'vhs'
    AND mc.copycondition = 'new'
    AND mc.movieid IN (SELECT mt.movieid FROM MOVIES.movietitle mt
WHERE
    mt.year < 1990
    AND mt.rating IN ('pg','r')
    AND mt.categoryid IN (SELECT mc.categoryid
FROM
    MOVIES.moviecategory mc
WHERE
    mc.rentalprice=(SELECT MAX (rentalprice)
FROM
    MOVIES.moviecategory
WHERE
    categoryid=mc.categoryid)))
    AND mr.CUSTOMERID=cs.CUSTOMERID
    AND ri.RENTALID=mr.RENTALID

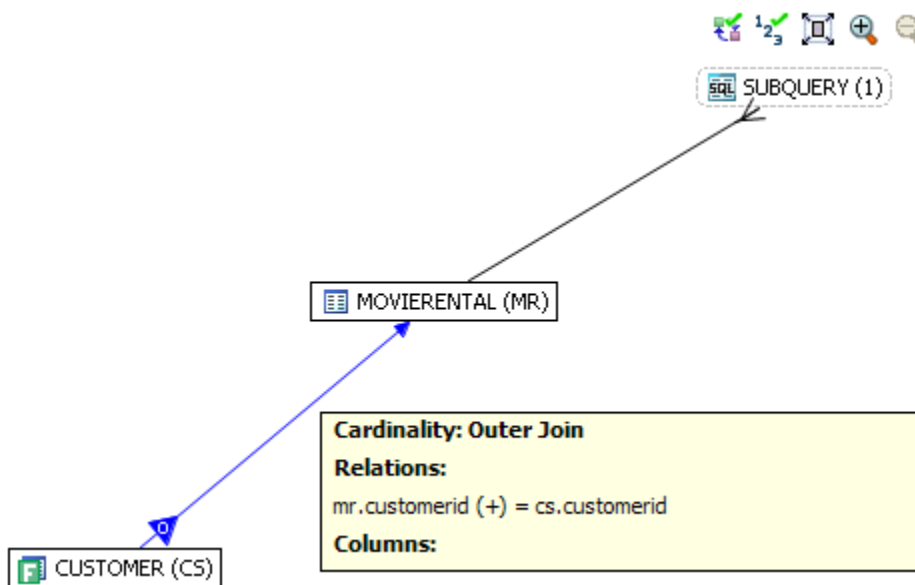
```

## Outer join

The bold SQL predicate in the statement below defines the outer join between customer and movierental.

```
select cs.*  
from MOVIES.customercs, MOVIES.movierentalmr  
where  
    length (cs.lastname) = 8 and cs.zip > 75062 and 1 < cs.customerid + 2 and cs.phone between 9625569900 and 9999569900 and mr.rentalid  
    = (select max (ri.rentalid)  
  
from MOVIES.rentalitem ri, MOVIES.moviecopy mc  
where  
    ri.itemnumber > 1 and mc.moviecopyid = 700) and mr.customerid(+) = cs.customerid;
```

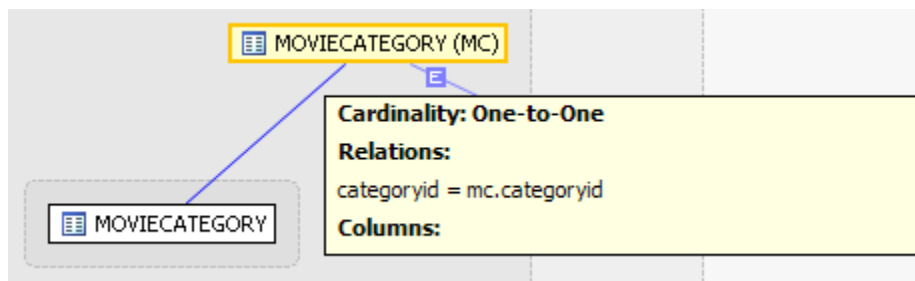
The following screen shot illustrates how the outer join is displayed in the VST diagram.



## Unique

The subquery below illustrates a unique relationship between two primary keys.

```
...select max(rentalprice) from MOVIES.moviecategory where categoryid = mc.categoryid...
```



## Not in or not exists join

The following SQL contains a NOT IN subquery (shown in bold below) that is graphically represented with the Subquery summary icon and the NOT IN join.

```
SELECT CS.*  
FROM
```

```
MOVIES.CUSTOMERCS,
MOVIES.MOVIERENTALMR
```

WHERE

```
CS.ZIP > '75062'
AND MR.RENTALID NOT IN (SELECT MAX ( MOVIES.BIG_STATEMENT5.CUSTOMERID)
```

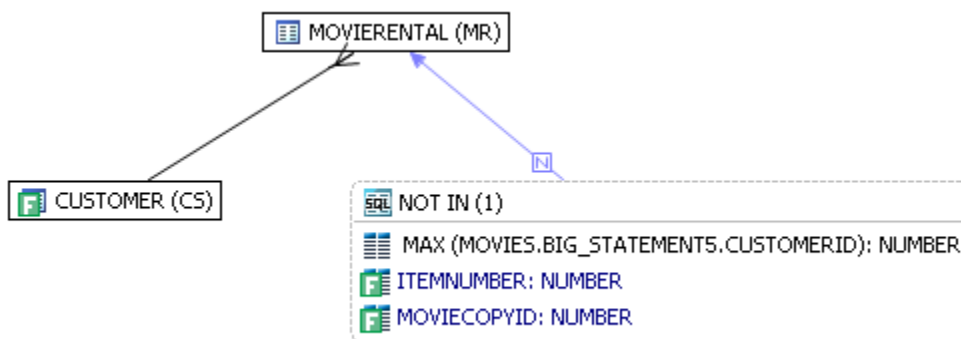
FROM

```
MOVIES.RENTALITEMRI,
MOVIES.MOVIECOPYMC,
MOVIES.BIG_STATEMENT5
```

WHERE

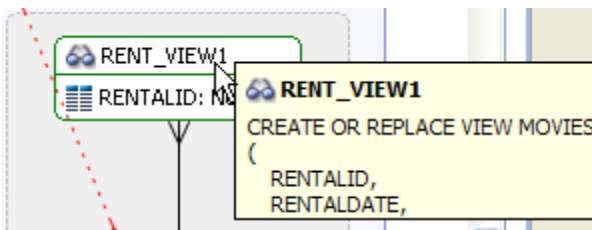
```
RI.ITEMNUMBER>1
AND MC.MOVIECOPYID=700)
AND MR.CUSTOMERID=CS.CUSTOMERID;
```

Graphically, this statement would look like this:



## Viewing object SQL

Hover over the name of an object to view the object SQL as shown in the diagram below.



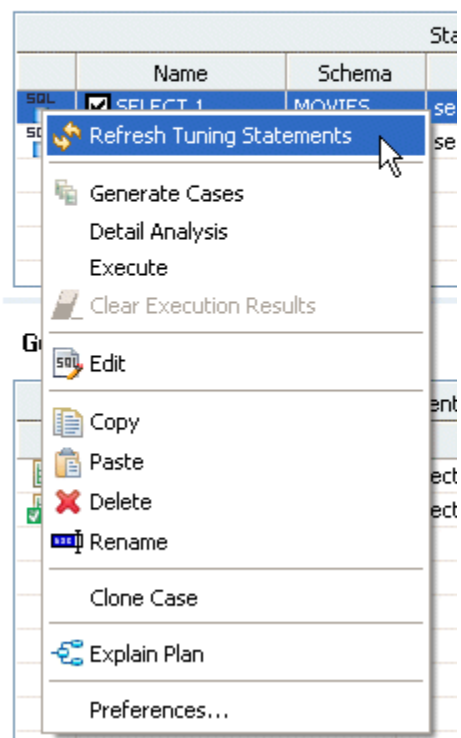
## Refreshing tuning statements

At times you may see an error on the Overview page, which when you mouse over it, indicates that the tuning statements are out of sync and need to be refreshed. This can happen, for example, if you tune a statement, then delete it, and insert another SQL query for tuning.

### To refresh the tuning statements

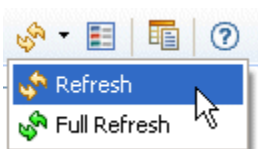
In the Tuning Statements area of the Overview tab, right-click the tuning statement and select **Refresh Tuning Statements**.

## Tuning Statements ☒ Generate ca:



## Refreshing the VST diagram

There are two refresh options available: **Refresh** and **Refresh All**. Click the **Refresh** list as shown below to gain access to these options.



- **Refresh:** Regenerates the Analysis tab including the VST diagram. Any changes made on the tab are reflected in the diagram.
- **Full Refresh:** Re-caches all objects used in (or related to) the query, then regenerates the Analysis tab including the VST diagram. This option is typically used when the underlying objects have been recently changed.