What is a master table?

Designing master tables is the first step in database design. Master tables hold basic information about a system. To understand master tables, we need to understand the concept of a transaction. A transaction is an activity performed by entities within the system. The permanent data is called MASTER data like Employee Information, department info. Day to day business data is called TRANSACTION data like sales, salaries, expenses. In SAP, Material, customer, and vendor etc. are called master data tables.

**EXISTS vs. IN**

The EXISTS function searches for the presence of a single row meeting the stated criteria as opposed to the IN statement which looks for all occurrences.

TABLE1 - 1000 rows

TABLE2 - 1000 rows

(A)

SELECT t1.id

FROM table1 t1

WHERE t1.code IN (SELECT t2.code

FROM table2 t2);

(B)

SELECT t1.id

FROM table1 t1

WHERE EXISTS (SELECT '1'

FROM table2 t2

WHERE t2.code = t1.code)

For query A, all rows in TABLE2 will be read for every row in TABLE1. The effect will be 1,000,000 rows read from items.

In the case of query B, a maximum of 1 row from TABLE2 will be read for each row of TABLE1, thus reducing the processing overhead of the statement.

**Rule of thumb:**

If the majority of the filtering criteria are in the subquery then the IN variation may be more performer.

If the majority of the filtering criteria are in the top query then the EXISTS variation may be more performer.

I would suggest, you should try both variants and see which works the best.

**See the link as details:**

<https://asktom.oracle.com/pls/asktom/f?p=100:11:0::no::p11_question_id:442029737684>

**Example 02:**

**SELECT DISTINCT** s.OFFICIALNUMBER **AS** O\_No , s.FULLNAME **AS Name**, r.RANK\_NAME **AS Rank**, bpu.NAME **AS** posting\_unit , s.SAILORID

**FROM** sailor s

**left join** bn\_rank r **on** r.RANK\_ID = s.RANKID

**LEFT JOIN** bn\_branch b **ON** s.BRANCHID = b.BRANCH\_ID

**LEFT JOIN** bn\_daogroup dao **ON** b.DAO\_GROUPID = dao.GROUP\_ID

**LEFT JOIN** bn\_posting\_unit bpu **ON** s.POSTINGUNITID = bpu.POSTING\_UNITID

**LEFT JOIN** transfer t **ON** s.SAILORID = t.SailorID

**LEFT JOIN** partii pa2 **on** pa2.PartIIID = s.FIRSTPARTID

**where** s.SAILORSTATUS=1

**AND** dao.GROUP\_ID **IN** (1,2,3,4,5,7)

**AND** p.ORG\_ID **IN** (63) **and not exists** (**select** 1 **FROM** sailor m **left join** assessment ass **on** ass.SailorID = m.SAILORID **where** AssessYear = **'2018' and** m.SAILORID = s.SAILORID )

**...or**

**not** *exists* (**select** 1 **FROM** assessment asses **where AssessYear** = **'2018' and** asses.**SailorID** = s.**SAILORID**)

In the above code, **not exists** and in the below code, **not in** are the same

**select** m.**SAILORSTATUS**,m.**SAILORID**, m.**OFFICIALNUMBER**, m.**FULLNAME**, (**CASE WHEN** (p.**Name** != **''**) **THEN** concat(r.**RANK\_NAME**, **'('**, p.**Name**, **')'**) **ELSE** r.**RANK\_NAME END**) **AS** RANK\_NAME, p.**NAME** posting\_unit

**from**

(**select SAILORSTATUS**, **SAILORID**, **OFFICIALNUMBER**, **FULLNAME**, **RANKID**, **FIRSTPARTID**, **POSTINGUNITID**, **BRANCHID**, **AREAID**, **ZONEID**

**from** sailor

**where SAILORSTATUS** = 1

**and SAILORID not in** (**select SailorID from** assessment **where AssessYear** = 2018))m

**left join** bn\_rank r **on** m.**RANKID** = r.**RANK\_ID**

**left join** bn\_branch br **on** br.**BRANCH\_ID** = m.**BRANCHID**

**LEFT JOIN** bn\_posting\_unit p **ON** m.**POSTINGUNITID** = p.**POSTING\_UNITID**

**left join** partii par **on** m.**FIRSTPARTID** = par.**PartIIID**

**LEFT JOIN** bn\_trade t **on** t.**TRADE\_ID** = par.**TradeID**

**LEFT JOIN** bn\_daogroup dao **ON** dao.**GROUP\_ID** = br.**DAO\_GROUPID**

**where** m.**SAILORSTATUS** = 1

**Example 03:**

SELECT 1 FROM TABLE\_NAME means, "Return 1 from the table". It is pretty unremarkable on its own, so normally it will be used with WHERE and often EXISTS. **select 1** from table will return the constant 1 for every row of the table. It's useful when you want to cheaply determine if record matches your where clause and/or join.

**SELECT \* FROM TABLE1 T1 WHERE EXISTS (**

**SELECT 1 FROM TABLE2 T2 WHERE T1.ID= T2.ID**

**);**

Basically, the above will return everything from table 1 which has a corresponding ID from table 2

**SELECT \* FROM TABLE1 T1 WHERE ID IN (SELECT ID FROM TABLE2);**

Sometimes **SELECT 1 FROM TABLE\_NAME** will be used. This does not offer significant benefit over selecting an individual column, but, depending on implementation, it may offer substantial gains over doing a **SELECT \***

If you mean something like this

**SELECT \* FROM AnotherTable**

**WHERE EXISTS (SELECT 1 FROM table WHERE...)**

Then it's a myth that the 1 is better than

**SELECT \* FROM AnotherTable**

**WHERE EXISTS (SELECT \* FROM table WHERE...)**

[The 1 or \* in the EXISTS is ignored and you can write this as per Page 191 of the ANSI SQL 1992 Standard:](http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt)

**SELECT \* FROM AnotherTable**

**WHERE EXISTS (SELECT 1/0 FROM table WHERE...)**

**Example 04:**

Another Example of **not in**

**RankID IN (SELECT bn\_rank.RANK\_ID FROM bn\_rank WHERE bn\_rank.EQUIVALANT\_RANKID IN ($branchRank))**

**Avoid In-Line Queries**

In-line queries execute for each column for each row.

For example, if a main query has 100 columns, and brings 1000 rows, then each column query executes 1000 times. Altogether, it is 100 multiplied by 1000 times. This is not scalable and cannot perform well. Avoid using in-line sub queries whenever possible.

Avoid the following use of in-line queries. If this query returns only a few rows this approach may work satisfactorily; however, if the query returns 10000 rows, then each sub or inline query executes 10000 times and the query would likely result in Stuck threads.