Challenging with Null values on column & indexing usage!

Does a index in Oracle maintained Null values?

Does Oracle use index when where clause contains "column Is null"?

There have always been issues with NULLs and indexes. The main issue being of course if the indexed columns are all null then the associated row is not indexed.
Generally, this is a good thing. If we have a table with lots of null values for indexed columns, then the associated rows are not indexed resulting in a smaller index structure. Also, very often we're simply not interested in result sets where the indexed values are null so it's generally not an issue.
However, what if the number of rows where the values are null are relatively small and what if we want to find all rows where the index column or columns are indeed null. If the column or columns don't have nulls indexed then a potentially expensive Full Table Scan (FTS) is the CBO's only option.
The first thing to point out is that nulls are actually indexed, if other columns in the index have a not null value. For example, if we have a concatenated index on columns (A,B), so long as A has a not null value then column B can have an indexed null value and if column B has a not null value then column A can have an indexed null value. Only if both columns A and B contain nulls, will the associated row not be indexed.
If column B has a NOT NULL constraint, then Oracle knows that B can not contain any null values. Therefore, if column A can contain null values, Oracle also knows that each and every null value of A must also be indexed as it's not possible to have an entirely null indexed entry.
Therefore, with an index on (A,B), we can use the index to return every null value for A, providing of course the CBO considers the costs of doing so to be cheaper than a FTS. We can also always of course use the index to return all null values of A for any corresponding not null value of B.
So with concatenated indexes and with at least one not null column, Oracle can guarantee that every null for all the other columns are contained within the index and so could potentially use the index for corresponding IS NULL predicates.
But what if the index has a single column or what if none of the indexes have a NOT NULL constraint, we're done for, the CBO won't be able to use the associated index to just retrieve nulls, right ?

As we know, provided an index scan is indeed faster than a full table scan.

Oracle does not put *tuples* that are all NULL into an index, therefor if you define an index on a single column e.g. supposed that user_id is a nullable field in person table.

CREATE INDEX idx_id ON person (user_id);

Insert some records into person table with some rows null in user_id;

Select * from person where user_id is Null;

that index will not contain rows where user_id is NULL.
As a result of that, the index cannot be used for a IS NULL condition.

But what happened when where clause contains IS NOT NULL?

select * from users where user_id is not null;

the optimizer will choose not to use the index (as it would do for any other query) if the selectivity is not high enough.

If the Nulls are few on the table, then fully scanning the table will be faster - or at least the optimizer thinks so.

Oracle **recommend** to usually create not null columns, and if you force to use nullable use below method:

If column contains many **nulls**, but queries often select all rows having a value, In this case, use the following phrase**:**
WHERE COL_X > -9.99 * power(10,125)

Using the preceding phrase is preferable to:
WHERE COL_X IS NOT NULL

Bad:
Select * from table
 WHERE COL_X IS NOT NULL;

Good:
Select * from table
WHERE COL_X > -9.99 * power(10,125);

Assuming that COL_X is a numeric column).

Other practical methods to handling Nulls:

However you can use a workaround if you need to do frequent IS NULL selections, by forcing the nulls to be indexed using a constant in the index definition:

CREATE INDEX idx_id ON person (user_id, 1).

That index will be used for a IS NULL condition

Another same solutions for barchar columns:

create index users_lastname on users_table (upper (lastname));

create index find_null on users_table (nvl (mycolumn, 'ISNULL');

Oracle would then create a psuedocolumn and create an index based on the pseudocolumn, making queries more efficient.

A practical and usefull methods in production to handling Nulls:

```
SELECT first_name, last_name
  FROM employees
 WHERE date_of_birth IS NULL;
```

Consider employee table has **SUBSIDIARY_ID not null and DATE_OF_BIRTH nullable fileds.**

the **EMP_DOB** index. It has only one column: the **DATE_OF_BIRTH**.
A row that does not have a **DATE_OF_BIRTH** value is not added to this index.

```
INSERT INTO employees (
subsidiary_id, employee_id, first_name, last_name, phone_number)
VALUES ( ?, ?, ?, ?, ? )
```

The insert statement does not set the **DATE_OF_BIRTH** so it defaults to **NULL**,
hence, the record is not added to the **EMP_DOB** index.
As a consequence, the index cannot support a query for records where **DATE_OF_BIRTH IS NULL**:

```
SELECT first_name, last_name
  FROM employees
 WHERE date_of_birth IS NULL;
```

Nevertheless, the record is inserted into a concatenated index if at least one index column is not **NULL**:

```
CREATE INDEX demo_null
ON employees (subsidiary_id, date_of_birth);
```

The above created row is added to the index because the **SUBSIDIARY_ID** is not **NULL**.

This index can thus support a query for all employees of a specific subsidiary that have no **DATE_OF_BIRTH** value:

```
SELECT first_name, last_name
  FROM employees
 WHERE subsidiary_id = ?
   AND date_of_birth IS NULL;
```

Please note that the index covers the entire where clause; all filters are used as access predicates during the **INDEX RANGE SCAN**.

So with consideration of leading column in index, you can force useing index when you have null values by create a composite index by adding not null's columns to your index plus nullable column.

demo_null index can be useful when you looking for only subsidiary_id **OR** when you looking for both subsidiary_id and date_of_birth**.**

Are you allowed to change the db structure?

If yes, in order to not have any WHERE column IS NULL or WHERE column IS NOT NULL condition in your queries, then fully normalize your tables (i.e. 5NF or 6NF) and make all columns that are used in conditions NOT NULL.

the null value has no place in the relational model, That's exactly what I meant. In theory that's all well. In practise though, I think it's rare to find a database without any NULLable column.

♨️ *fully* normalize.
A table in 1NF is normalized.
A table is fully normalized only if it is in 5NF or 6NF.
A table in 6NF will have no nullable columns.

That would allow you to make the field non-nullable, which increases performance somewhat, and works better with indexes.

🐘Postgres vs Oracle🔴

Can't you create an index on the boolean expression field IS NULL?

Yes, in Oracle you can not create index on a nullable column.

Coming from **PostgreSQL**, where I'm using those expression for indexing IS NULL, and since 9.0 leverages indexes for IS NOT NULL by default, it seems weird to me that "the all mighty" Oracle won't allow to use indexes for IS NULL.

♨️Postgres partial index

```
# create index idx_num on test(num) where num is not null;
```

In Postgres:

NULL values are included in (default) B-tree indexes since version Postgres 8.3.

Partial indexing is particularly useful for finding MAX()/MIN() values in indexes that contain many null values.

GIN indexes followed later:
As of PostgreSQL 9.1, null key values can be included in the index.

Deep inside of indexing and Null values:

The next thing to check would possibly be to use the DUMP function to again see what Oracle is likely to do with NULL values.

The DUMP function displays the raw decimal representation of the specific character (depending of course on the character-set) .

For NULL values however, there's actually nothing to display other than a NULL text to represent there's nothing actually there.
The best place to check of course is within the actual index itself. By determining the actual block that stores our example index, we can perform an index block dump and look at the resultant trace file that describes a representation of the index block to see precisely how Oracle deals with NULLs within indexes.
A quick check of the HEADER_FILE and HEADER_BLOCK in DBA_SEGMENTS will give us the index segment header location.To find the associated index root/leaf block simply add 1 to the HEADER_BLOCK.

Dump the block via the
ALTER SYSTEM DUMP DATAFILE a BLOCK b
command and look at the trace file in USER_DUMP_DEST (where '**a**' represents the datafile id and '**b**' represents the block id determined from dba_segments).

The resultant output clearly shows that yes:
1. Leading column NULLs values are all grouped together
2. They are all listed at the "end" of the index structure
3. Any NULLs in the non-leading indexed columns are listed "last" for each distinct value in the leading columns in which they appear
4. Any index entry consisting of nothing but NULLs are not actually stored within the index

Best Regards,
Alireza Kamrani
Senior RDBMS Consultant