Lab Yellowcom case study

Populate DB

- After the physical desing, before implementing any index on a specific table for a specific non-key field (according to the result of the physical design), we need to populate the database in order to see if the DBMS will use the indexes that we will implement
- With reference to the Yellowcom case study, it could be interesting to populate the following relations:
  - Customer
  - Contract
  - …

Populate Customer

```
CREATE OR REPLACE PROCEDURE populateCustomer as
BEGIN
  FOR i IN 1..400000  LOOP
    INSERT INTO Customer VALUES (
      dbms_random.string('L',16),
      dbms_random.string('L',10),
      dbms_random.string('L',10),
(SELECT TO_DATE(
TRUNC(DBMS_RANDOM.VALUE(
TO_CHAR(DATE '1910-01-01','J'),
TO_CHAR(DATE '1999-12-31', 'J'))), 'J') FROM DUAL));
  END LOOP;
END;
```

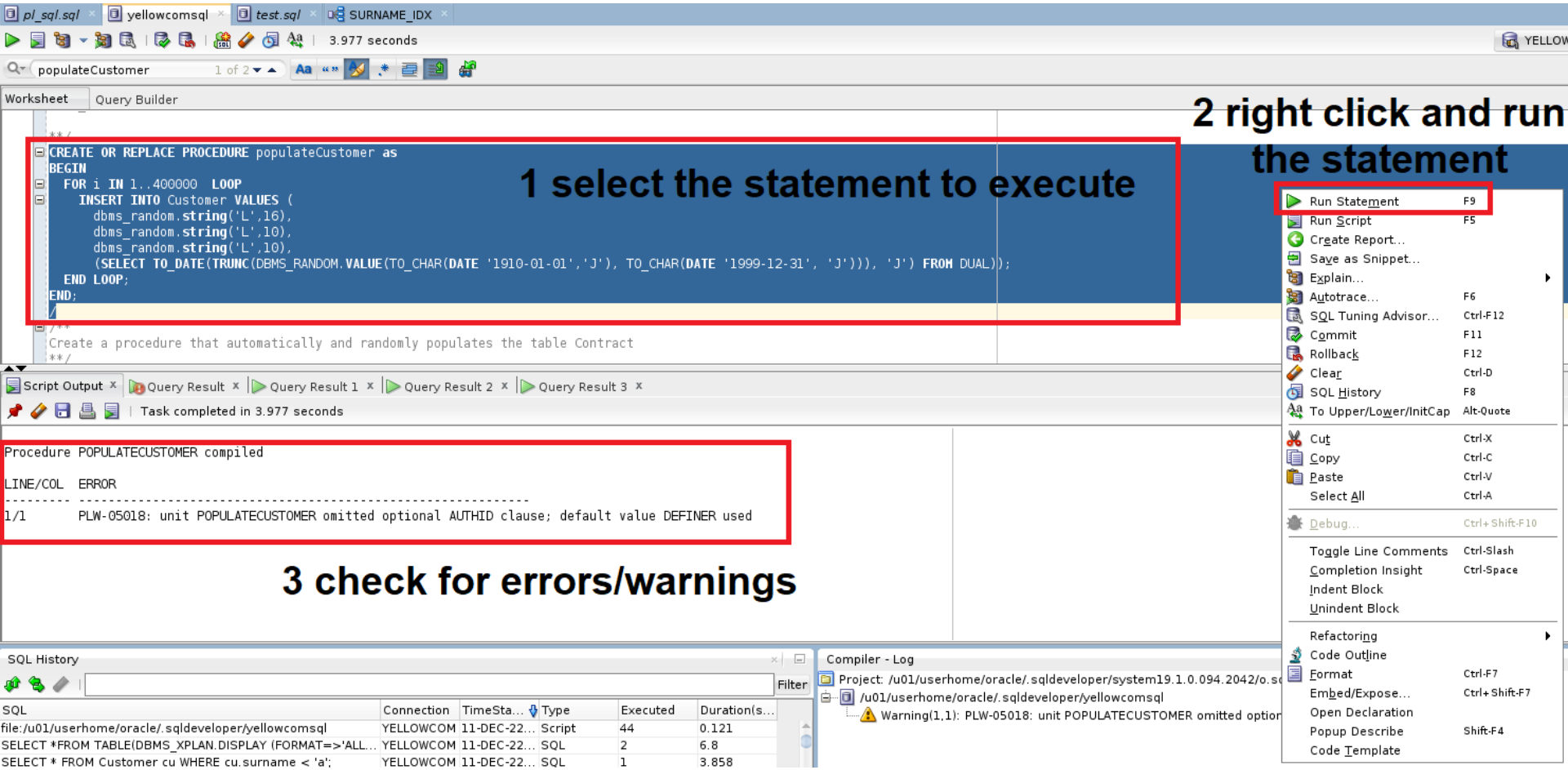
TO_CHAR(value, format_mask)
value : a number or date that will be converted to a string.
format_mask. Optional. This is the format that will be used to convert value to a string. 'J' stands for Julian date

*The **TRUNC(date)** function returns date with the time portion of the day truncated to the unit specified by the format model fmt.*

***DBMS_RANDOM.VALUE(low, high)** // for number and date*

Julian Date Calender
(PERPETUAL)

Day	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Day
1	001	032	060	091	121	152	182	213	244	274	305	335	1
2	002	033	061	092	122	153	183	214	245	275	306	336	2
3	003	034	062	093	123	154	184	215	246	276	307	337	3
4	004	035	063	094	124	155	185	216	247	277	308	338	4
5	005	036	064	095	125	156	186	217	248	278	309	339	5
6	006	037	065	096	126	157	187	218	249	279	310	340	6
7	007	038	066	097	127	158	188	219	250	280	311	341	7
8	008	039	067	098	128	159	189	220	251	281	312	342	8
9	009	040	068	099	129	160	190	221	252	282	313	343	9
10	010	041	069	100	130	161	191	222	253	283	314	344	10
11	011	042	070	101	131	162	192	223	254	284	315	345	11
12	012	043	071	102	132	163	193	224	255	285	316	346	12
13	013	044	072	103	133	164	194	225	256	286	317	347	13
14	014	045	073	104	134	165	195	226	257	287	318	348	14
15	015	046	074	105	135	166	196	227	258	288	319	349	15
16	016	047	075	106	136	167	197	228	259	289	320	350	16
17	017	048	076	107	137	168	198	229	260	290	321	351	17
18	018	049	077	108	138	169	199	230	261	291	322	352	18
19	019	050	078	109	139	170	200	231	262	292	323	353	19
20	020	051	079	110	140	171	201	232	263	293	324	354	20
21	021	052	080	111	141	172	202	233	264	294	325	355	21
22	022	053	081	112	142	173	203	234	265	295	326	356	22
23	023	054	082	113	143	174	204	235	266	296	327	357	23
24	024	055	083	114	144	175	205	236	267	297	328	358	24
25	025	056	084	115	145	176	206	237	268	298	329	359	25
26	026	057	085	116	146	177	207	238	269	299	330	360	26
27	027	058	086	117	147	178	208	239	270	300	331	361	27
28	028	059	087	118	148	179	209	240	271	301	332	362	28
29	029		088	119	149	180	210	241	272	302	333	363	29
30	030		089	120	150	181	211	242	273	303	334	364	30
31	031		090		151		212	243		304		365	31



The warning could be avoided by using explicit clause AUTHID CURRENT_USER
CREATE OR REPLACE PROCEDURE populateCustomer **`AUTHID CURRENT_USER AS`** …

After compiling:
EXECUTE populateCustomer

SELECT (*) FROM CUSTOMER

Populate Contract

```
CREATE OR REPLACE PROCEDURE populateContract AS
BEGIN
  FOR i IN 1..50 LOOP
    INSERT INTO Contract VALUES (
      RechargeableType(
      DBMS_RANDOM.STRING('U', 10),
      (SELECT * FROM (SELECT REF(T) FROM Customer T ORDER BY dbms_random.value)
WHERE rownum < 2),
      (SELECT (TRUNC(dbms_random.value(1, 10),9)*1000000000) FROM dual),
      (SELECT * FROM (SELECT REF(T) FROM TariffPlan T ORDER BY dbms_random.value)
WHERE rownum < 2),
      (SELECT (TRUNC(dbms_random.value(1, 10),2)*10) FROM dual)));
    END LOOP;
END;
```

CREATE INDEX

Purpose

Use the CREATE INDEX statement to create an index on:

- One or more columns of a table, a partitioned table, an index-organized table, or a cluster
- One or more scalar typed object attributes of a table or a cluster
- A nested table storage table for indexing a nested table column

An **index** is a schema object that contains an entry for each value that appears in the indexed column(s) of the table or cluster and provides direct, fast access to rows. Oracle Database supports several types of index:

- Normal indexes. By default, Oracle Database creates B-tree indexes (B+tree)
- **Bitmap indexes**, which store rowids associated with a key value as a bitmap (useful for DW)
- **Partitioned indexes**, which consist of partitions containing an entry for each value that appears in the indexed column(s) of the table
- **Function-based indexes**, which are based on expressions. They enable you to construct queries that evaluate the value returned by an expression, which in turn may include built-in or user-defined functions.
- **Domain indexes**, which are instances of an application-specific index of type *indextype*

Print Customers Operation

Print all the customers in alphabetically ascending order w.r.t. the surname

```
SELECT surname FROM customer ORDER BY surname ASC;
```

An index could be defined for the surname since it is not a key attribute for the relation and it is not already defined.

```
CREATE INDEX  surname_idx ON customer(surname)
```

Is the index used by the DBMS?

```
EXPLAIN PLAN

SET STATEMENT_ID = 'customer_order_by_surname_asc'

FOR SELECT surname FROM customer ORDER BY surname ASC;



SELECT operation, options, object_name

   FROM plan_table WHERE statement_id = 'customer_order_by_surname_asc'

  ORDER BY id;
```

Is the index used by the DBMS?

```
EXPLAIN PLAN

SET STATEMENT_ID = 'customer_order_by_surname_asc'

FOR SELECT surname
FROM customer
ORDER BY surname ASC;

SELECT operation, options, object_name

    FROM plan_table

    WHERE statement_id = 'customer_order_by_surname_asc'

   ORDER BY id;
```

Script Output | Query Result | Query Result 1 | Query Result 2 | Query Result 3

SQL | All Rows Fetched: 3 in 0.1 seconds

	OPERATION	OPTIONS	OBJECT_NAME
1	SELECT STATEMENT	(null)	(null)
2	SORT	ORDER BY	(null)
3	INDEX	FAST FULL SCAN	SURNAME_IDX

You can see if the DBMS use effectively the user-defined index

Is the index used by the DBMS?

Print all the customers within a range

```
EXPLAIN PLAN

SET STATEMENT_ID = 'customer_<a'

FOR SELECT * FROM Customer cu WHERE cu.surname < 'a' ;

SELECT operation, options, object_name

   FROM plan_table

   WHERE statement_id = 'customer_<a'

  ORDER BY id;
```

Is the index used by the DBMS?

```
EXPLAIN PLAN

SET STATEMENT_ID = 'customer_<a'

FOR SELECT * FROM Customer cu WHERE cu.surname < 'a' ;

SELECT operation, options, object_name

    FROM plan_table

    WHERE statement_id = 'customer_<a'

   ORDER BY id;
```

ript Output | Query Result | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4

SQL | All Rows Fetched: 3 in 1.137 seconds

	OPERATION	OPTIONS	OBJECT_NAME
1	SELECT STATEMENT	(null)	(null)
2	TABLE ACCESS	BY INDEX ROWID BATCHED	CUSTOMER
3	INDEX	RANGE SCAN	SURNAME_IDX

Is the index used by the DBMS?

```
EXPLAIN PLAN

SET STATEMENT_ID = 'customer_<a'

FOR SELECT * FROM Customer cu WHERE cu.surname < 'a' ;

SELECT operation, options, object_name, id, parent_id

   FROM plan_table

   WHERE statement_id = 'customer_<a'

  ORDER BY id;
```

How to read the plan

```
EXPLAIN PLAN

SET STATEMENT_ID = 'customer_<a'

FOR SELECT * FROM Customer cu WHERE cu.surname < 'a' ;

SELECT operation, options, object_name, id, parent_id

    FROM plan_table

    WHERE statement_id = 'customer_<a'

   ORDER BY id;
```

cript Output x | Query Result x | Query Result 1 x | Query Result 2 x | Que

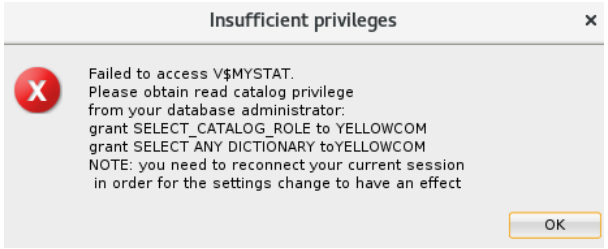
SQL | All Rows Fetched: 3 in 1.977 seconds

	OPERATION	OPTIONS	OBJECT_NAME	ID	PARENT_ID
1	SELECT STATEMENT	(null)	(null)	0	(null)
2	TABLE ACCESS	BY INDEX ROWID BATCHED	CUSTOMER	1	0
3	INDEX	RANGE SCAN	SURNAME_IDX	2	1

Each line in the plan is a separate operation. These operations are linked via a parent/child relationship:

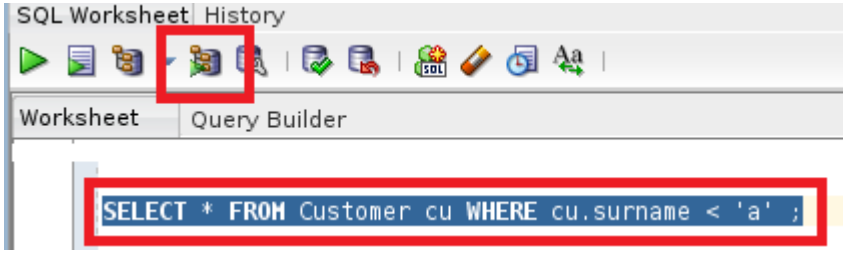
- the SELECT statement at the top is the root
- the tables are the leaves at the bottom
  - Accessed via the b+tree index and a range scan of the connected leaves
- in between you'll find a whole host of possible operations
  - Table access through the primary key rowid by exploiting another index (automatically generated by the DBMS)

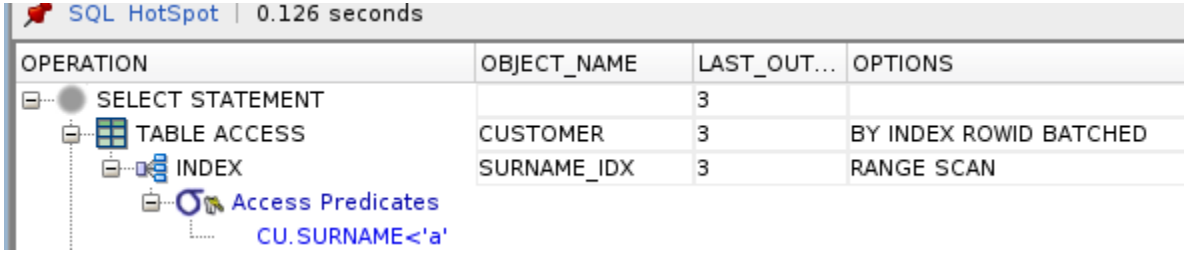
Autotrace



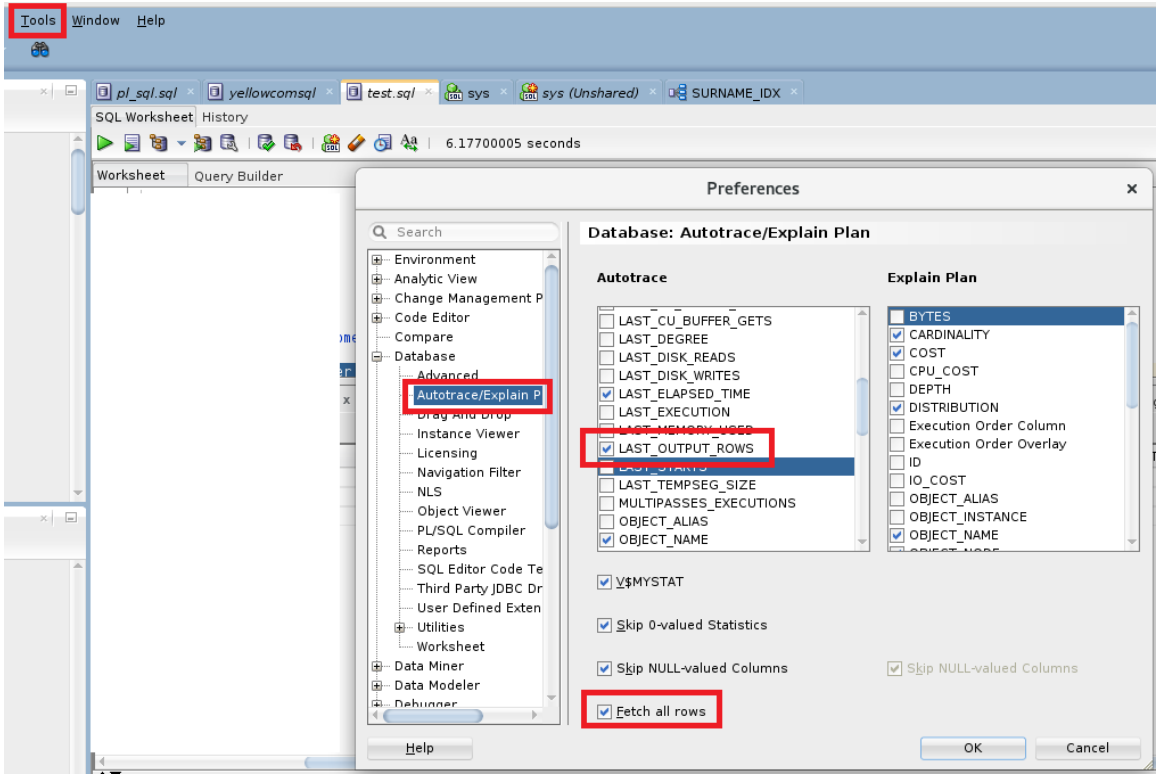
Connect with sys or system and execute with sys or system or enable yellowcom to do so:

SET AUTOTRACE ON
GRANT SELECT_CATALOG_ROLE TO YELLOWCOM
GRANT SELECT ANY DICTIONARY TO YELLOWCOM





How many rows are involved?



AUTOTRACE for the query: SELECT * FROM Customer cu WHERE cu.surname < 'a' ;

SQL HotSpot | 0.126 seconds

OPERATION	OBJECT_NAME	LAST_OUT...	OPTIONS
SELECT STATEMENT		3	
TABLE ACCESS	CUSTOMER	3	BY INDEX ROWID BATCHED
INDEX	SURNAME_IDX	3	RANGE SCAN
Access Predicates			
CU.SURNAME<'a'			

Only three tuples are ordered before ‘a’ character.
Note that uppercase comes first in order of lowercase ‘a’

Other types of useful indexes for the case study

Bitmap Join Indexes. A bitmap join index is a bitmap index for the join of two or more tables. For each value in a table column, the index stores the rowid of the corresponding row in the indexed table. A bitmap join index is an **efficient means of reducing the volume of data that must be joined by performing restrictions in advance**. For an example of when a bitmap join index would be useful, assume that users often query the number of employees with a particular job type.

```
SELECT COUNT(*) FROM employees, jobs WHERE  employees.job_id = jobs.job_id AND    jobs.job_title = 'Accountant';
```

The preceding query would typically use an index on jobs.job_title to retrieve the rows for *Accountant* and then the **job ID**, and an index on **employees.job_id** to find the matching rows. *To retrieve the data from the index itself rather than from a scan of the tables*, you could create a bitmap join index as follows:

```
CREATE BITMAP INDEX employees_bm_idx ON employees (jobs.job_title) FROM employees, jobs WHERE
employees.job_id = jobs.job_id;
```

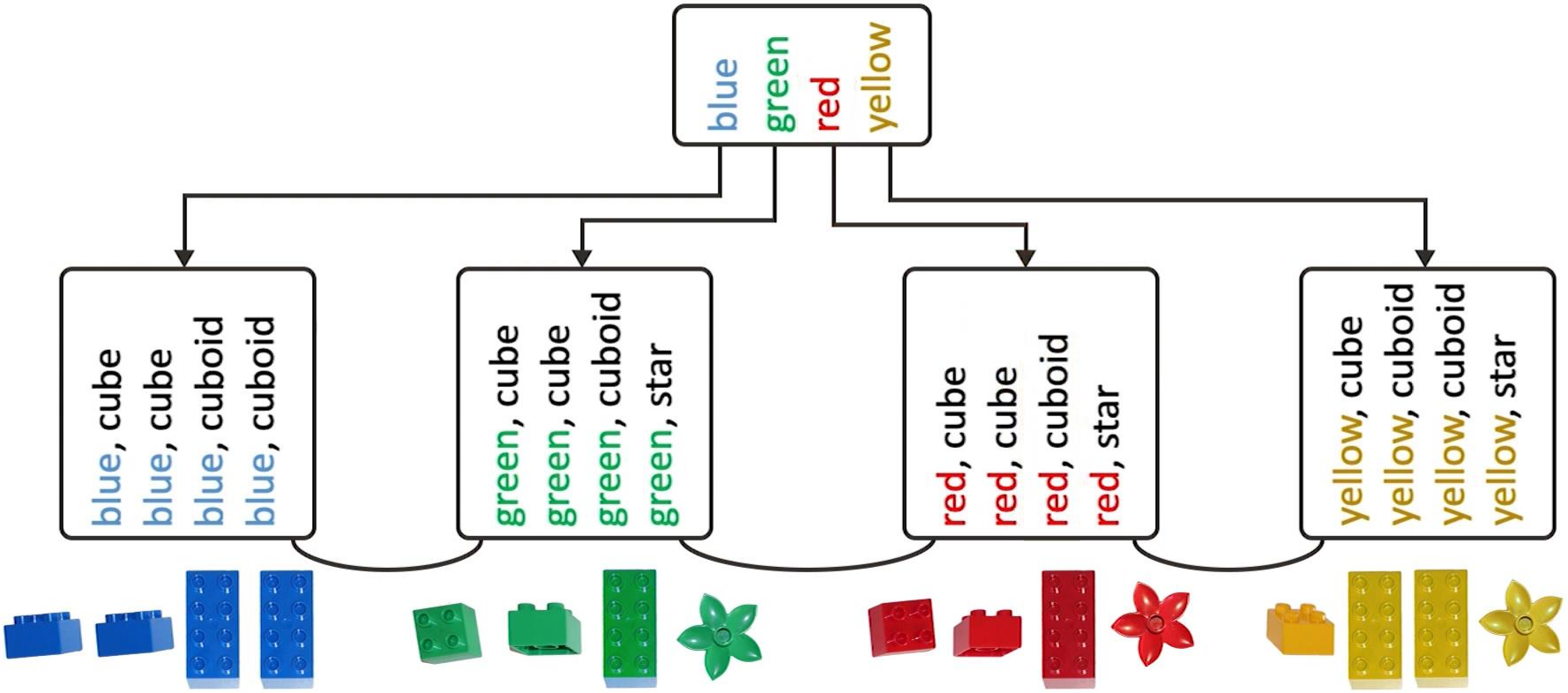
With reference to Yellowcom Case Study, a possible application could be implemented via a BITMAP INDEX on customer, contract. An example could be the following.

```
CREATE BITMAP INDEX customer_bm_idx ON customer(contract) FROM customer, contract
WHERE  customer.id = contract.customer;
```

```
SELECT COUNT(*) FROM customer, contract WHERE customer.id = contract.customer AND
contract.codecontract = 'a_specific_contract_to_retrieve'
```

Indexes on multiple columns

```
create index brick_colour_shape_i
on bricks ( colour, shape );
```



How to organize the columns?

1. Prefer first the columns appearing with equality conditions (=) within the clause WHERE
2. Followed by < (>) conditions
3. Followed by *select_order_by* columns

References

P. Atzeni, S. Ceri, P. Fraternali, S. & R. Paraboschi Torlone
Databases: Architectures and lines of evolution. Second edition.
McGraw-Hill Books Italy, 2007
Chapter 1

For more information:

Navathe, Elmasri
Of Database Systems - Fundamentals, fourth / fifth edition
Pearson Education, 2004