Increase Efficiency of Relational Databases Using Instruments of Second Normal Form

Conference Paper · December 2021

DOI: 10.1109/ATIT54053.2021.9678605

5 authors, including:

Oleksandr Rolik
National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”
44 PUBLICATIONS **133** CITATIONS
SEE PROFILE

2021 IEEE Smart Information Systems and Technologies (SIST)
28-30 April, 2021, Nur-Sultan, Kazakhstan

Increase efficiency of relational databases using instruments of Second Normal Form

Oleksandr Rolik, Kseniia Ulianytska, Maryna Khmeliuk, Volodymyr Khmeliuk, Uliana Kolomiiets
Department of Automation and Control in Technical Systems, National Technical University of Ukraine
“Igor Sikorsky Kyiv Polytechnic Institute”, 03056
o.rolik@kpi.ua, k.ulianytska@kpi.ua, marburdey@gmail.com, Hmelyuk@gmail.com,
uliankacolombo@gmail.com

***Abstract*—Supporting and administrating relational databases is one of the important stages of application lifecycle. This article explains the difference between Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP) systems and, basically, represents one more method to increase efficiency of storing data and changing logical model for improving executing SQL (Structured Query Language) constructions. Not only the task of designing relational databases is the subject of scientific research, because there are many problems appears till application is in use. The article focuses on the normalization of databases and to use Normal Form in area of efficiency OLTP based application.**

***Keywords*—Relational Database, Relational Theory, Normalization, Normal Forms, Compound keys, Increase efficiency**

I. Introduction

In the world of informational technologies - a big question, devoting to retrieving data from databases appears for software developers and database administrators. No matter how beautiful and modern will be design of application, if queries executes more than a second, all application will be useless.

On the stage of supporting applications, based on relational databases, efficient of executing SQL-queries goes on the first plan.

Database technology is at the center of many types of information systems. Among these information systems are decision support systems and executive information systems. OLTP environments use database technology to transact and query against data, and support the daily operational needs of the business enterprise [1]. OLAP environments use database technology to support analysis and mining of long data horizons, and to provide decision makers with a platform from which to generate decision making information. [1]

Applications built on OLTP principals demand increasing efficiency of executing storing procedures and each SQL-query in particular. That’s why we have many methods to upgrade and rewrite SQL-query itself or to make relational database management system (RDBMS) works productive and effective.

This article devoted to reviewed one of Codd’s Normal Form [3]. Based on original definition of Second Normal Form [4] new method of increasing efficiency will be represented for applications, based on OLTP principals.

II. Problem Statement

There are exist two different ways to construct any relational database: OLTP and OLAP.

OLTP database structures are characterized by storage of “atomic” values, meaning individual fields cannot store multi-values. They are also transaction oriented as the name implies [1]. Applications build on that principles have to be very fast for response. Nobody will wait loading some forms of application more, than a few seconds, until query executes.

Alternatively, OLAP database structures are generally aggregated and summarized. There is an analytical orientation to the nature of the data, and the values represent a historical view of the entity [1]. According to Terr [2], real-time data warehousing combines realtime activity with OLAP concepts. He discusses the problem with latency and differentiates between “realtime” and “near real-time” systems. The business goal of using real-time data analysis is further supported by Terr [2] in his discussion on the compelling reasons for building real-time data analysis systems.

For OLTP databases extremely important to be fast to response and to use operative and hard memory more efficiency. Analyzing Second Normal Form in previous article [3] one very important conclusion was made. This conclusion concerns and memory storage and logical model, which has influence for executing SQL-queries in particular.

In this article will be shown how to transform on of the basic statement of relational theory from the area of modeling relational database into area of analyzing and increasing efficiency of relational database usage for SQL-queries in applications.

To study this issue, traditionally, a number of relational databases were designed and students conducted laboratory studies. All tables are normalized to 3NF. In most tables, a composite primary key is missing. Where there was a need for unique combinations of attributes, the integrity constraint of the unique key was applied. This article will explain how to use composite primary in the future for increasing efficiency of RDBMS and SQL query executing.

III. METHODS TO INCREASE THE EFFICIENCY OF SQL QUERIES

On the way of building application, obviously, database model should be written. Developers make their choice of database type (relational model or NoSQL model) and choice of database management system (DBMS). After it, based on UML model, using Use Case and User Stories models SQL query should be written.

Any SQL query can be written in multiple ways but we should follow best practices to achieve better query performance. Well known practices gathered in list below [6], excluding parts for particular DBMS:

- Use EXISTS instead of IN to check existence of data.
- Avoid * in SELECT statement. Give the name of columns which you require.
- Choose appropriate Data Type. E.g. To store STRING-type values use varchar in place of text data type. Use text data type, whenever you need to store large data (more than 8000 characters).
- Avoid NCHAR and NVARCHAR if possible since both the data types takes just double memory as CHAR and VARCHAR.
- Avoid NULL in fixed-length field. In case of requirement of NULL, use variable-length (VARCHAR) field that takes less space for NULL.
- Avoid HAVING Clause. HAVING clause is required if you further wish to filter the result of an aggregations.
- Drop unused Indexes.
- Better to create indexes on columns that have integer values instead of characters. Integer values use less overhead than character values.
- Use JOINs instead of sub-queries.
- Use WHERE expressions to limit the size of result tables that are created with joins.
- Use WITH (NOLOCK) while querying the data from any table.
- Use SET NOCOUNT ON and use TRY-CATCH to avoid deadlock condition.
- Avoid CURSORs since CURSOR are very slow in performance.
- Use UNION ALL in place of UNION if possible.
- Use Schema name before SQL objects name.
- Use Stored Procedure for frequently used data and more complex queries.
- Keep transaction as small as possible since transaction lock the processing tables data and may results into deadlocks.
- Avoid Multi-statement Table Valued Functions (TVFs). Multi-statement TVFs are more costly than inline TVFs.

It is well known, that adequate normalized model with full infological analysis produced efficient structure of database for executing query of any difficulty. But even ideal database needs optimization in time. Below (Fig.1) is represented abstract overview of query optimization in relational DBMS:

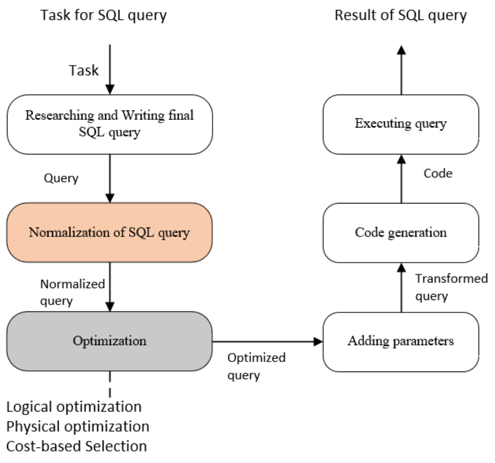


Figure 1. Query optimization plan

Fig.1 represents general approach, but the basic steps in query optimization on the physical optimization are following [7]:

1. Query Plan generation
2. Cost Estimation

a. Enumerate reasonable alternative plans

b. Estimate the properties of input using catalog and also estimate the properties of result

c. Repeat for space

3. Choose plan
4. Intimate Catalog Manager

The goal of a query optimizer is to find a good evaluation plan for a given query. Optimizing a relational algebra expression involves two basic steps:

1. Enumerating alternative plans for evaluating the expression

2. Estimating the cost of each enumerated plan, and choosing the plan with the least estimated cost.

On Fig.2 is shown physical optimization plan:

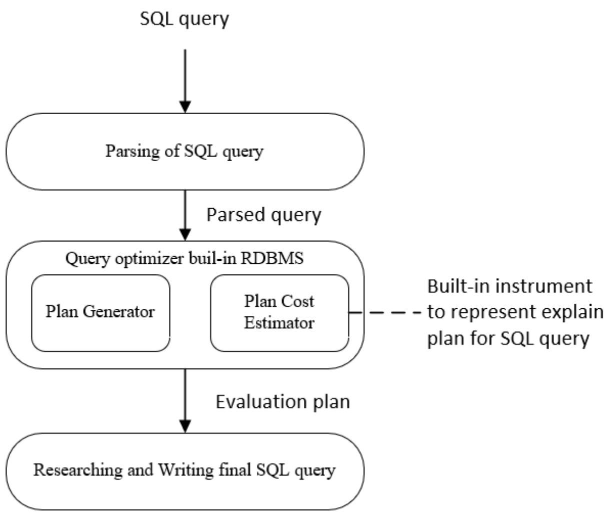

SQL query
Parsing of SQL query
Parsed query
Query optimizer buil-in RDBMS
Plan Generator
Plan Cost Estimator
Built-in instrument to represent explain plan for SQL query
Evaluation plan
Researching and Writing final SQL query

Figure 2. Query physical optimization plan (parsing, optimization and executing)

These are the basic and well-known steps to optimize and in it's turns to increase efficiency of SQL query.

Now we have to look carefully on Fig.1 on block "Normalization of SQL query". This article will show how to increase efficiency of executing some particular query using one of the well-known Normal Form – Second Normal Form.

IV. METHOD BASED ON SECOND NORMAL FORM FOR OLTP

According to Codd [5], a relation is in the second normal form if it fulfills the following two requirements: relation should be in first normal form and it does not have any non-prime attribute that is functionally dependent on any proper subset of any candidate key of the relation. A non-prime attribute of a relation is an attribute that is not a part of any candidate key of the relation.

After detailed analysis in paper [3] was shown, that Second Normal Form and Boyce and Codd Normal Form haven't been reviewed for many years. On the practice integral primary key, which is not logically connected to relation is used. Most of logical data model includes tables as it shown on small relational schema on Figure 3:

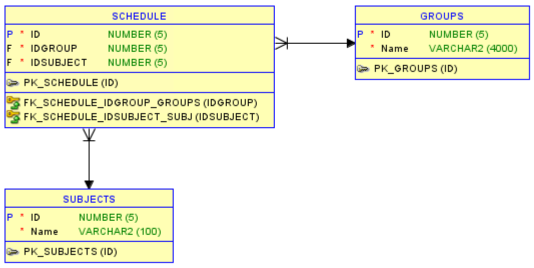


Figure 3. Relational data model for "Schedule at the university" subject domain

We have ID – primary key in each table, which is automatically generated and not logically connected to relational table.

In paper [3] after reviewing a modernized definition for Second Normal Form was proposed: relation is in 2NF if it is in 1NF and as for primary key it uses integral non-prime for relation attribute with automatic generation of the following value.

Nevertheless, compound primary key can be used in the area of optimization executing query and increasing efficiency of relational DBMS in particular.

For table "Schedule" in model on Fig.3 we can delete attribute ID and make compound primary key, using IDGROUP and IDSUBJECT. Logically, combination of values for these two attributes will never repeat, that's why we can use them as unique primary key. Profit from this action:

- reducing usage of memory for values of one attribute – integral primary key by using existing non-key attribute as a potential primary key;
- reducing usage of indexes – instead of 3 indexes on different attribute, there will be only one index for compound primary key.

V. LITERATURE REVIEW IN THE RESEARCH FIELD

A lot of information about optimization of SQL queries exists [6-8, 10], but not very often optimization connected with Normal Forms and normalization as itself.

There are materials related to the analysis of specific normal forms [14, 17], functional dependencies [15] and keys (complex primary key, foreign key) in particular [16]. In this material described terms, methods, and design questions of relational theory. All of them complement and do not contradict each other and all basics terms are unshakable.

However, there are some articles, where we can find hints of future changing of relational theory. In [16] among other things, this paper includes a discussion of the pros and cons of surrogate keys, which leads to revision of, for example, 2NF.

An extensive and detailed treatment of the problems caused by duplicates. The discussion of duplicates directly connected to integrity constraints such as primary, foreign, unique keys [14].

As it was shown in [3], working with complex primary key is very difficult. Complex primary keys are used in a very specific way, they are not being used as a foreign key in another tables. But this article practically represented the way how to successfully implement compound primary into relational schema.

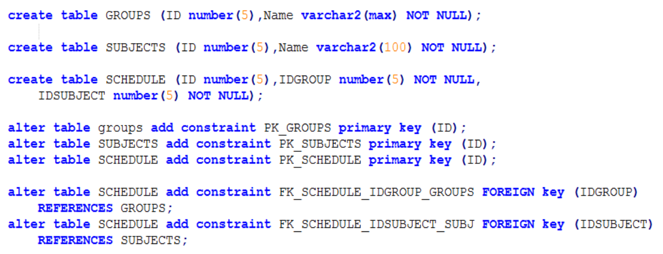
Modern relational DBMS (such as Oracle and MS SQL Server) do not allow several primary keys in a table, so functional dependencies between primary keys are impossible by default.

For increasing efficiency of executing query in time with several condition can be applied the way of adding compound primary key.

VI. Experiment and Results

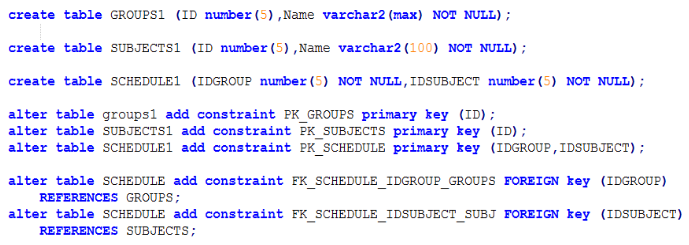
On the stage of experiment relational tables were implemented using model on Fig.3.

Using DDL (data definition language) statements structure of tables were added to relational DBMS Oracle Database 11g XE (Fig.4, Fig.5). There were created two schemas with two different models (using integral and compound primary key):



```
create table GROUPS (ID number(5),Name varchar2(max) NOT NULL);

create table SUBJECTS (ID number(5),Name varchar2(100) NOT NULL);

create table SCHEDULE (ID number(5),IDGROUP number(5) NOT NULL,
    IDSUBJECT number(5) NOT NULL);

alter table groups add constraint PK_GROUPS primary key (ID);
alter table SUBJECTS add constraint PK_SUBJECTS primary key (ID);
alter table SCHEDULE add constraint PK_SCHEDULE primary key (ID);

alter table SCHEDULE add constraint FK_SCHEDULE_IDGROUP_GROUPS FOREIGN key (IDGROUP)
    REFERENCES GROUPS;
alter table SCHEDULE add constraint FK_SCHEDULE_IDSUBJECT_SUBJ FOREIGN key (IDSUBJECT)
    REFERENCES SUBJECTS;
```

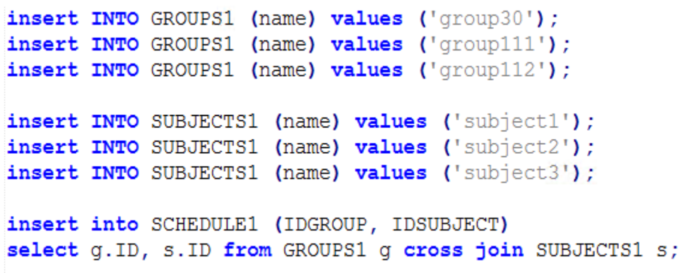
Figure 4. DDL scripts for creating structure with integral primary key



```
create table GROUPS1 (ID number(5),Name varchar2(max) NOT NULL);

create table SUBJECTS1 (ID number(5),Name varchar2(100) NOT NULL);

create table SCHEDULE1 (IDGROUP number(5) NOT NULL,IDSUBJECT number(5) NOT NULL);

alter table groups1 add constraint PK_GROUPS primary key (ID);
alter table SUBJECTS1 add constraint PK_SUBJECTS primary key (ID);
alter table SCHEDULE1 add constraint PK_SCHEDULE primary key (IDGROUP,IDSUBJECT);

alter table SCHEDULE add constraint FK_SCHEDULE_IDGROUP_GROUPS FOREIGN key (IDGROUP)
    REFERENCES GROUPS;
alter table SCHEDULE add constraint FK_SCHEDULE_IDSUBJECT_SUBJ FOREIGN key (IDSUBJECT)
    REFERENCES SUBJECTS;
```

Figure 5. DDL scripts for creating structure with compound primary key

For each table using DML scripts for insertion were added test data. Example of DML insertion represented on Fig.6:



```
insert INTO GROUPS1 (name) values ('group30');
insert INTO GROUPS1 (name) values ('group111');
insert INTO GROUPS1 (name) values ('group112');

insert INTO SUBJECTS1 (name) values ('subject1');
insert INTO SUBJECTS1 (name) values ('subject2');
insert INTO SUBJECTS1 (name) values ('subject3');

insert into SCHEDULE1 (IDGROUP, IDSUBJECT)
select g.ID, s.ID from GROUPS1 g cross join SUBJECTS1 s;
```

Figure 6. DML scripts for inserting test data

In first schema we have table with tree attributes – SCHEDULE (ID, IDGROUP, IDSUBJECT):

- integral primary key with mandatory index;
- two foreign keys with optional indexes, but indexes were added, because these attributes will be often to use in queries.

In second schema we have table with two attributes – SCHEDULE (IDGROUP, IDSUBJECT):

- compound primary key with mandatory index on both attributes.

In fact, in memory these two tables take an equal space (Fig.7):

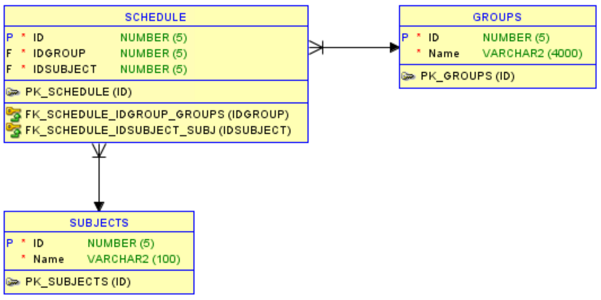
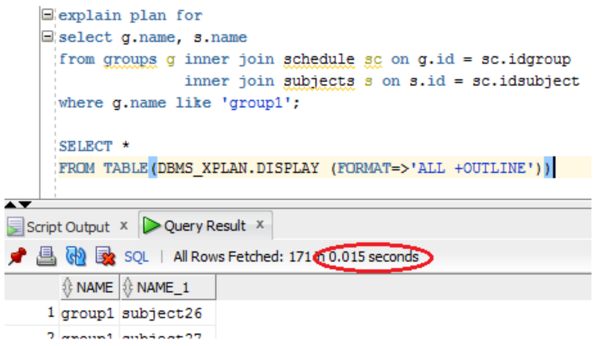


Figure 7. DBA_SEGMENTS for experimental tables

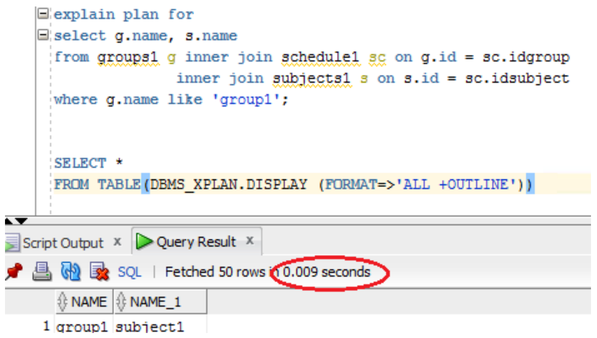
But for executing query, the second schema gave us a better result (Fig.8 compare with Fig.9):



```
explain plan for
select g.name, s.name
from groups g inner join schedule sc on g.id = sc.idgroup
              inner join subjects s on s.id = sc.idsubject
where g.name like 'group1';

SELECT *
FROM TABLE(DBMS_XPLAN.DISPLAY (FORMAT=>'ALL +OUTLINE'))
```

```
All Rows Fetched: 171 in 0.015 seconds
  NAME   NAME_1
1 group1 subject26
```

Figure 8. Executing query with integral primary key



```
explain plan for
select g.name, s.name
from groups1 g inner join schedule1 sc on g.id = sc.idgroup
              inner join subjects1 s on s.id = sc.idsubject
where g.name like 'group1';


SELECT *
FROM TABLE(DBMS_XPLAN.DISPLAY (FORMAT=>'ALL +OUTLINE'))
```

```
Fetched 50 rows in 0.009 seconds
  NAME   NAME_1
1 group1 subject1
```

Figure 9. Executing query with compound primary key

Besides, in the first case we have All Rows Fetched, but in the second – Fetched particular number it means, that first query searched among all records, and the second – only among fetched, which increased efficiency almost in two times.

VII. Conclusions

Designing relational database at first, we should follow all necessary normal forms and principals of modern relational DBMS. Some of these principals built on integral primary keys not logically connected with relational tables, all key attributes must be unified – with the same data type and with the same integrity constraint to avoid anomalies and syntaxis mistakes in the future.

Absolutely not all tables with two foreign keys can be rebuild using compound primary keys, because more often we have situation with additional attributes and combination of foreign keys might be duplicated. So, before transforming relational tables using Second Normal Form principals, we should analyze our subject domain and go to data-logical or even info-logical stage of design.

Forbidden to transform relational tables with one primary key into tables with compound primary key, if one or more tables references on it. If do that, we will get redundant information including indexes and more attributes in referencing tables.

Further research will be devoted to optimization of OLTP systems using normalization principals. Also, it should be compared with OLAP systems, as so this system contains large massive with data without any referencing on time executing SQL queries.

References

[1] *OLTP and OLAP Data Integration: A Review of Feasible Implementation Methods and Architectures for Real Time Data Analysis.*

[2] Terr, S., *Real-Time Data Warehousing: Real-Time Data Warehousing 101*, DM Review Online. Retrieved July 5, 2004. (http://www.dmreview.com/article_sub.cfm?articleId=7524)

[3] O.Rolik, O.Amons, K.Ulianytska, V.Kolesnik: *Modernization of the Second Normal Form and Boyce-Codd Normal Form for relational theory.* - ICCSEEA 2020: Advances in Computer Science for Engineering and Education III, pp. 296-305.

[4] Codd E.F., *Normalized Data Base Structure: A Brief Tutorial*//Proc. of 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control. - N.-Y.: ACM, pp.1-17.

[5] Codd E.F., *Further Normalization of the Data base Relational Model* // Data Base Systems. - N.J.: Prentice-Hall, 1972, pp.33-64.

[6] *25 tips to Improve SQL Query Performance*. (https://www.winwire.com/25-tips-to-improve-sql-query-performance/)

[7] Sibtain Masih, *“Query Optimization”,* - Techmight. (https://techmightsolutions.blogspot.com/2014/07/query-optimization.html)

[8] Navita Kumari, *SQL Server Query Optimization Techniques - Tips for Writing Efficient and Faster Queries.* - International Journal of Scientific and Research Publications, Volume 2, Issue 6, June 2012.

[9] Jiyuan Shi, *Research and Practice of SQL Optimization in ORACLE.* - 2010 Third International Symposium on Information Processing.

[10] K. Thomas, *ORACLE expert Advanced Programming*, 2002.

[11] J. Ketchen and G. Hult, *Bridging organization theory and supply chain management: The case of best value supply chains,* vol. 25, no. 2, pp. 573-580, 2006.

[12] Y. Alotaibia, B. Ramadan, *A Novel Normalization Forms for Relational Database Design throughout Matching Related Data Attribute.* - Published Online September 2017 in MECS. - I.J.Engineering and Manufacturing, pp. 65-72.

[13] Fagin R. A., *Normal Form for Relational Databases That is Based on Domains and Keys// ACM Transactions on Database Systems*.- 1981.- V.6, #.3.- pp. 387-415.

[14] Calenko M., *Modeling semantics in databases.* - M.: Science, 1988.

[15] E. F. Codd, *Relational Completeness of Data Base Sublanguages*, in Randall J. Rustin (ed.), Data Base Systems, Courant Computer Science Symposia Series 6. Englewood Cliffs, N.J.: Prentice Hall, 1972.

[16] E. F. Codd, *Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks,* IBM Research Report RJ599, August 19th, 1969.

[17] E. F. Codd and C. J. Date, *Much Ado about Nothing,* in C. J. Date, Relational Database Writings 1991–1994. Reading, Mass.: Addison-Wesley, 1995.

[18] Date C.J., *SQL and Relational Theory: How to Write Accurate SQL Code (2nd edition).* - O’Reilly Media, Inc., 2012. – p.446.

[19] Y. Alotaibia, B. Ramadan, *A Novel Normalization Forms for Relational Database Design throughout Matching Related Data Attribute.* - Published Online September 2017 in MECS. - I.J.Engineering and Manufacturing, pp. 65-72.