
LUT
University
LAND
OF THE
CURIOUS

TABLE OF CONTENTS

CT60A7650 – DATABASE SYSTEMS MANAGEMENT

APPLICATION PERFORMANCE

Lecture

Jiri Musto, D.Sc.

APPLICATION CODE AND SQL

- Majority of performance problems come from
  - Poorly coded programs
  - Improperly coded SQL

- Either the SQL/code was bad to begin with or
- SQL/code was fine but the amount of data, usage or access changes reduced the performance of the original part

DESIGNING APPLICATIONS FOR RELATIONAL ACCESS

Design choice	Issue
Type of SQL	Is the SQL being used correct for this application?
Programming language	Is the programming language optimized and capable?
Transaction design and processing	Are the transactions properly designed and does the program use them appropriately?
Locking strategy	Are wrong types of locks being used or held too long?
COMMIT strategy	Is SQL COMMIT being used to minimize locking?
Batch processing	Are batch programs designed appropriately?
Online processing	Are online applications designed to minimize the amount of information returned?

OPTIMIZER AND OPTIMIZATION

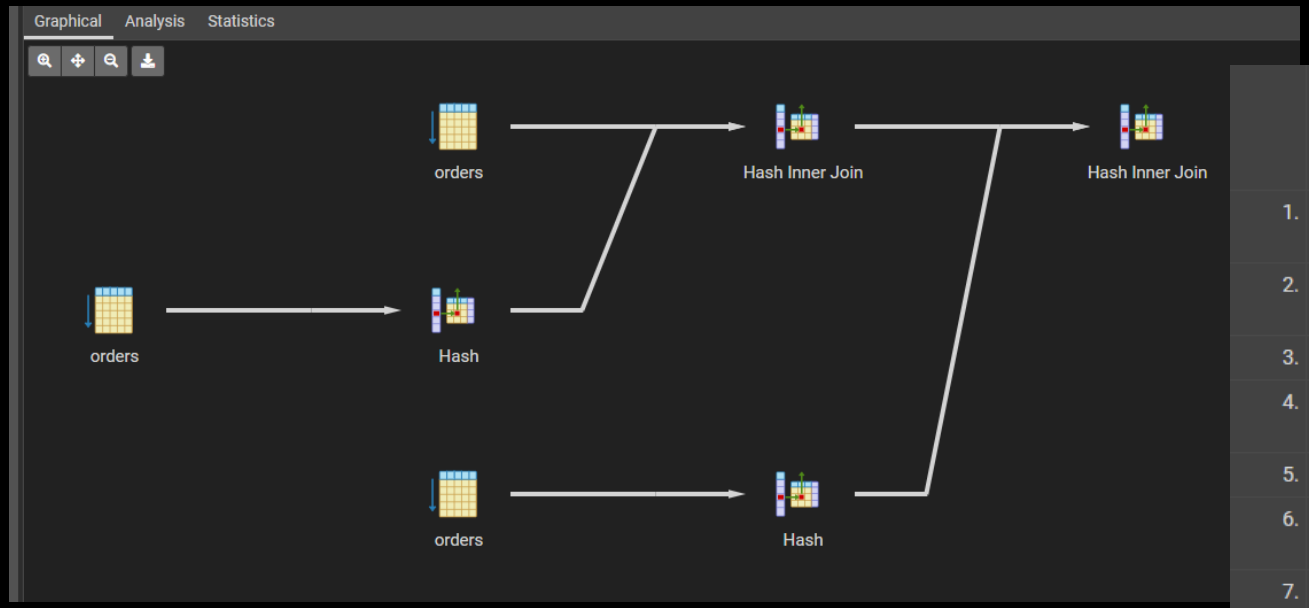
- Know how the DBMS optimizer works
- Application developer defines ***what*** data is needed
- DBMS defines ***where*** data is located
- Optimizer defines ***how*** to navigate the database efficiently
- Remember:
  - Optimizers are great but not perfect
  - Poorly designed SQL queries can only be optimized to a certain level

TESTING YOUR DATABASE

- Using a test database that has less data vs. production database
  - Faster to test
  - Statistics may not match production
- Using a copy of production database
  - Statistics match the production
  - May take more time to test

QUERY ANALYSIS

- DBMS offer tools to analyse queries
- The tools can help identify potential issues with the query



	Node	Rows Actual	Loops
1.	→ Hash Inner Join (rows=10712 loops=1) **Hash Cond**: (o2.custid = o3.custid)	10712	
2.	→ Hash Inner Join (rows=830 loops=1) **Hash Cond**: (o1.orderid = o2.orderid)	830	
3.	→ Seq Scan on orders as o1 (rows=830 loops=1)	830	
4.	→ Hash (rows=830 loops=1) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 132 kB	830	
5.	→ Seq Scan on orders as o2 (rows=830 loops=1)	830	
6.	→ Hash (rows=830 loops=1) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 132 kB	830	
7.	→ Seq Scan on orders as o3 (rows=830 loops=1)	830	

WHY AN INDEX WAS NOT CHOSEN?

- Search argument?
  - No search argument, no index
- Joining large number of tables?
  - An unpredictable query may be produced with a large number of tables
- Statistics are current?
  - Large number of changes has been made to the database, may need a manual update
- Stored procedures?
  - May need to recompile or reoptimize
- Additional predicates needed?
  - Different WHERE condition may enable the optimizer to operate properly

EXAMPLE STATISTICS

Query Editor | Query History | Scratch Pad

```
1  SELECT * FROM pg_stats WHERE tablename = 'orders';
```

Data Output | Explain | Messages | Notifications

	schemaname name	tablename name	attname name	inherited boolean	null_frac real	avg_width integer	n_distinct real	most_common_vals anyarray	most_common_freqs real[]
1	public	orders	empid	false	0	4	9	{4,3,1,8,2,7,6,9,5}	
2	public	orders	shipaddress	false	0	21	-0.29036143	{"Kirchgasse 1234","7890 Suffolk Ln.","Kirchgasse 9012","8901 Suffolk Ln.","Taucherstraße 2345","7890 Milton Dr.","9012 Suffolk Ln.","Taucherstraße...	.0.004819277,0.004819277,0.004819277,0.004819277,0.004819277,0.004819277,0.004819277,0.004819277,0.004
3	public	orders	shipregion	false	0.61084336	4	19	{SP,RJ,ID,OR,"Co. Cork",WA,NM,Táchira,BC,Lara,Essex,Québec,"Nueva Esparta",AK,"Isle of Wight",WY,CA,MT,DF}	{0.059036143
4	public	orders	shipcountry	false	0	6	21	{Germany,USA,Brazil,France,UK,Venezuela,Austria,Sweden,Canada,Italy,Mexico,Spain,Finland,Belgium,Ireland,Denmark,Switzerland,Argentina,Port...	{0.14698794,0.14698794
5	public	orders	orderid	false	0	4	-1	[null]	
6	public	orders	custid	false	0	4	-0.10722891	{71,20,63,24,37,5,35,65,9,25,44,87,10,34,39,41,46,89,4,51,62,47,66,76,7,32,67,83,11,30,38,49,55,56,59,68,80,84,86,31,61,72,75,88,14,19,28,48,3,6,21,...	3156,0.0072289156,0.006024096,0.006024096,0.006024096,0.006024096,0.006024096,0.006024096,0.006024096,0.00602409
7	public	orders	orderdate	false	0	8	-0.57831323	{"2008-02-26 00:00:00","2008-03-03 00:00:00","2008-03-06 00:00:00","2008-03-11 00:00:00","2008-03-16 00:00:00","2008-03-19 00:00:00","2008-03-...	0.0036144578,0.0024096386,0.0024096386,0.0024096386,0.0024096386,0.0024096386,0.0024096386,0.002409
8	public	orders	requireddate	false	0	8	-0.54698795	{"2008-03-26 00:00:00","2008-04-28 00:00:00","2007-02-13 00:00:00","2008-03-24 00:00:00","2008-04-08 00:00:00","2008-04-17 00:00:00","2008-04-...	0.0036144578,0.0036144578,0.0036144578,0.0036144578,0.0036144578,0.0036144578,0.0036144578,0.0036144578,0.003614
9	public	orders	shippeddate	false	0.025301205	8	-0.46626505	{"2008-04-10 00:00:00","2008-03-18 00:00:00","2008-01-23 00:00:00","2008-01-30 00:00:00","2008-04-08 00:00:00","2008-04-24 00:00:00","2008-05-...	0.0036144578,0.0036144578,0.0036144578,0.0036144578,0.0036144578,0.0036144578,0.0036144578,0.0036144578,0.003614
10	public	orders	shipperid	false	0	4	3	{2,3,1}	
11	public	orders	freight	false	0	8	-0.9626506	{"0,56 €","0,75 €","1,15 €","1,21 €","1,27 €","1,35 €","1,36 €","1,93 €","3,01 €","3,26 €","3,52 €","4,41 €","4,87 €","4,93 €","4,98 €","4,99 €","6,27 €","8,12 €","12,...	0.0024096386,0.0024096386,0.0024096386,0.0024096386,0.0024096386,0.0024096386,0.0024096386,0.0024096386,0.002409
12	public	orders	shipname	false	0	14	-0.29036143	{"Destination CUVPF","Ship to 71-A","Destination RVDMF","Ship to 63-B","Ship to 71-B","Ship to 63-A","Ship to 63-C","Ship to 65-A","Ship to 71-C","Desti...	.0.004819277,0.004819277,0.004819277,0.004819277,0.004819277,0.004819277,0.004819277,0.004819277,0.004
13	public	orders	shipcity	false	0	9	70	{"Rio de Janeiro",London,Boise,"Sao Paulo",Graz,Cunewalde,"México D.F.",Bräcke,Cork,Albuquerque,Luleå,"San Cristóbal",Marseille,"Buenos Aires","F...	3374,0.009638554,0.009638554,0.008433735,0.008433735,0.008433735,0.008433735,0.008433735,0.008433735,0.007228915
14	public	orders	shippostalc...	false	0	6	-0.29036143	{10159,10305,10157,10280,10306,10279,10281,10285,10307,10169,10222,10267,10368,10166,10197,10199,10202,10208,10209,10231,10294,1...	.0.004819277,0.004819277,0.004819277,0.004819277,0.004819277,0.004819277,0.004819277,0.004819277,0.004
15	public	orders	shippedbef...	false	1	1	0	[null]	

QUERY TWEAKING

- Disable access paths for a query by tweaking the WHERE condition
- This may lead to the usage of different indices
- Consider this: there is and index based on ***salary***

```
SELECT    last_name, first_name, empno, deptno
FROM      employee
WHERE     empno BETWEEN '001000' AND '009999'
AND       (salary > 50000.00 OR 0 = 1)
ORDER by last_name;
```

SQL CODING AND TUNING FOR EFFICIENCY

1. Identify data requirements
2. Ensure that required data is available
3. Translate the requirements into SQL
4. Test the SQL for accuracy and results
5. Review access paths for performance
6. Tweak SQL for better access paths
7. Optimize the SQL query
8. Repeat steps 4 to 7 until performance is acceptable
9. Repeat 8 when problems arise
10. Repeat everything when needs change

ADDITIONAL TUNING TIPS

- Create indices when necessary
- When possible, do not perform arithmetic in SQL
- Use SQL functions & procedures
- Build proper constraints
- Beware the impact of triggers and cascading commands

CT60A7650 – DATABASE SYSTEMS MANAGEMENT

DATA INTEGRITY

Lecture

Jiri Musto, D.Sc.

TYPES OF INTEGRITY

- Structural integrity (database)
  - Database objects are created, formatted and maintained properly
  - Errors in system and connecting applications may cause structural faults
  - For example: Program fails during runtime and inserted data is corrupted
- Semantic integrity (data)
  - Correct meaning of data and relationships between data are maintained
  - Loose checks and integrity controls in the database may cause semantics to fail
  - For example: A 2 year-old is marked as married in a database

STRUCTURAL INTEGRITY ISSUES

- If structural integrity is not maintained, database access will be compromised
- Pointer corruptions
  - If the pointers do not point to the correct data, accessing data is difficult
  - Index, XML, large text data, etc.
- Page header corruption
  - If header is corrupted, data within the page may be inaccessible
- Backup file issues
  - Incorrect format, wrong location of data
  - Cannot be used for recovery

MANAGING STRUCTURAL PROBLEMS

- Use DBMS utility programs
  - dbcc (SQL Server)
  - CHECK/REPAIR (DB2)
  - pg_checksum, pg_amcheck, (PostgreSQL)
  - etc.
- DBMS offer different utilities for different operations
  - Checking if indices work
  - Checking if pages are corrupted
  - Checking memory usage
  - And many others

SEMANTIC INTEGRITY

Form of integrity	Description	Examples
Entity integrity	Unique entities	Primary keys
Uniqueness	Unique values	UNIQUE constraints
Data types	Data type and length. Can be user defined	char, string, integer, Boolean
Default values	Assign default value if user does not give one	DEFAULT constraint
Check constraints	Given value is checked before insert or update. Can use comparison or complex checks	CHECK constraint, NOT NULL
Triggers	Procedure that is automatically run when an event happens	CREATE TRIGGER
Referential integrity	Data is correct in all corresponding tables	Foreign keys

DATA CONSTRAINTS

- Data type
  - Normal data types include CHAR, INT, VARCHAR etc.
  - User can define their own data types as well
- Default
  - Only one default value per column
  - Needs to be supported by the column, can be null
- Check
  - Create rules the data value has to follow
  - Can be simple '*larger than*' or complex '*in specified values*' rules
  - Outperforms most other data integrity methods

```
CREATE DOMAIN contact_name AS
    VARCHAR NOT NULL CHECK (value !~ '\s');
```

```
CHECK (relationship IN ('married', 'single', 'unknown')
```

NULL VALUES

- NULL can mean multiple things
  - Missing, unknown, arbitrary, not applicable
- Define what you mean by NULL
- How to handle NULLs
  - Different DBMS handle NULLs differently
  - Don’t allow NULLs by using NOT NULL
  - Assign DEFAULT values
  - Use NULLs carefully

	NULL/0	one/zero	zero/zero
Ingres 6.4/03	NULL	float point err no data	float point err no data
Oracle 6.0	NULL	divide by 0 err no data	divide by 0 err no data
Progress 6.2	NULL	NULL	NULL
R:BASE 4.0a	NULL	divide by 0 err no data	divide by 0 err no data
Rdb	truncation at runtime divide by 0	truncation at runtime divide by 0	truncation by runtime divide by 0
SQL Server 4.2	NULL	NULL & error	NULL & error
SQL Base 5.1	NULL	plus infinity	plus infinity
Sybase 4.9	NULL	Null & error	NULL & error
WATCOMSQL	NULL	NULL	NULL
XDD 2.41	NULL	divide by 0 err no data	divide by 0 err no data

PROCEDURES

- Procedures are user-defined functions that can be manually called
  - Can give arguments to procedures
- Procedures support transactions
- PostgreSQL added support in version 11
- Advantages
  - Reusable, can do different updates with minor tweaks
  - Faster than always writing new SQL queries
  - Secure
- Disadvantages
  - Specific to DBMS
  - Debugging
  - Dependency

```
CREATE PROCEDURE insert_data(a integer, b integer)
LANGUAGE SQL
AS $$
INSERT INTO tbl VALUES (a);
INSERT INTO tbl VALUES (b);
$$;

CALL insert_data(1, 2);
```

FUNCTIONS

- User defined functions that can be called manually or nested in SQL queries
  - Can give arguments to functions
  - Can be used to return values if necessary
- PostgreSQL uses functions to create triggers
- Similar benefits and drawbacks as procedures

```
CREATE OR REPLACE FUNCTION totalRecords ()
RETURNS integer AS $total$
declare
    total integer;
BEGIN
   SELECT count(*) into total FROM COMPANY;
   RETURN total;
END;
$total$ LANGUAGE plpgsql;
```

PROCEDURE VS. FUNCTION (IN POSTGRESQL)

Procedure	Function
Can use transactions	Cannot use transactions
Cannot return values (with exceptions)	Can return values
Manually called	Can be added to triggers
Cannot be nested in other commands	Can be nested in other commands (triggers, SELECT, INSERT etc.)

TRIGGERS

- Event-driven that are automatically run
  - Cannot be explicitly run
- DBMS offer different ways to create triggers
  - PostgreSQL requires functions for triggers
- Tied to database modifications
  - Insert, update, delete
  - Before, after, instead of
  - For each row / statement
  - Reference OLD/NEW

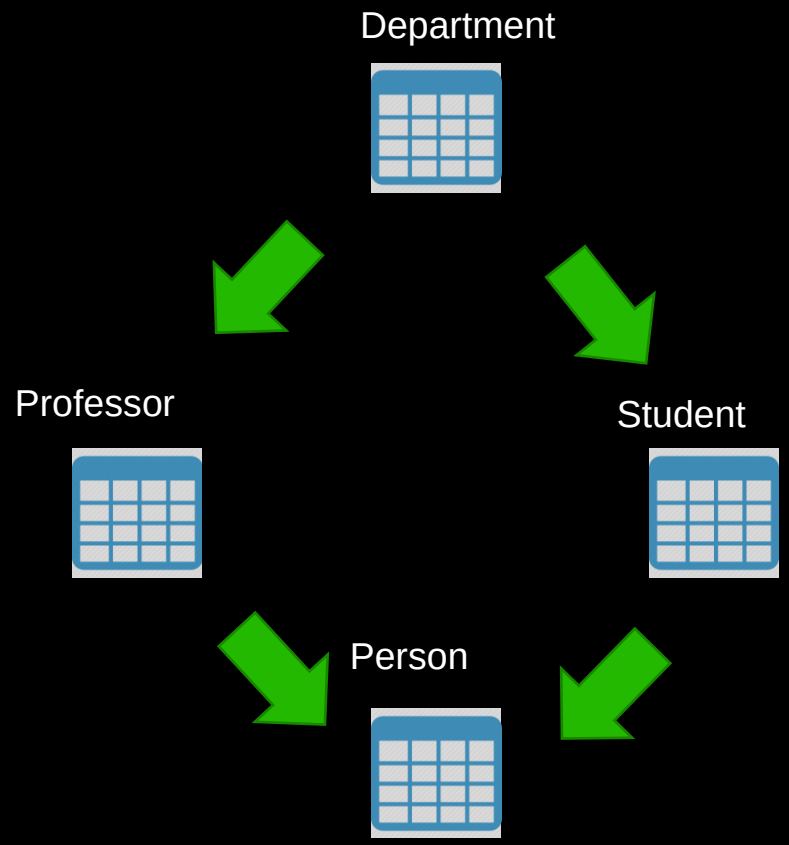
MANAGING TRIGGERS

- Triggers can contain INSERT, UPDATE and DELETE statements
  - Can have other triggers tied to the commands
  - So called ‘nested trigger’
  - DBMS can have limits to the amount of nested triggers
- Useful for managing database integrity
  - If new data is inserted and another table should be updated, triggers can help

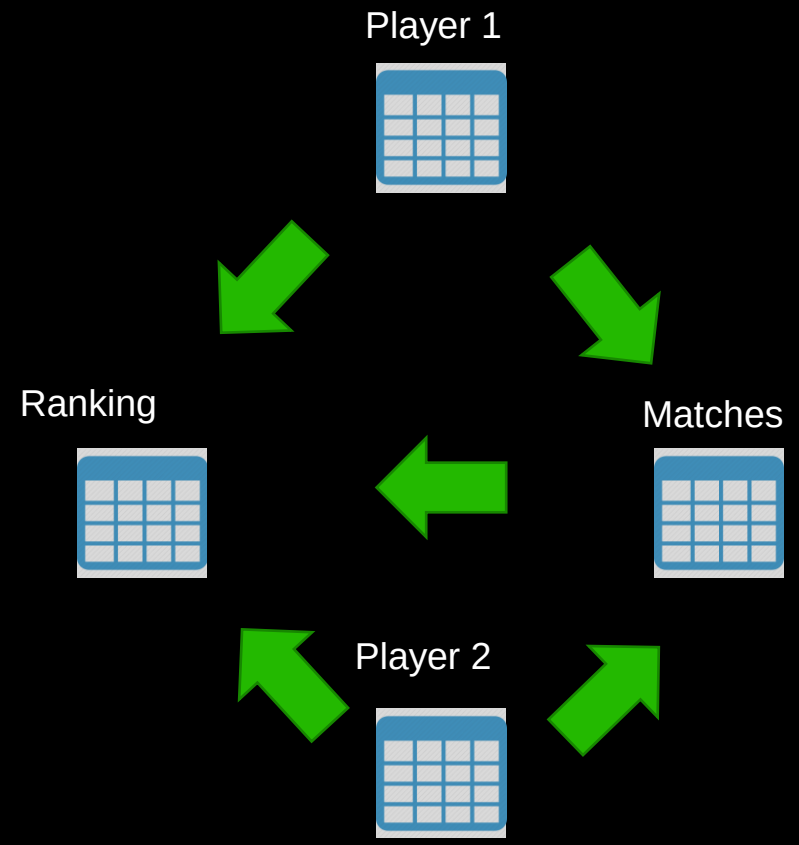
REFERENTIAL INTEGRITY

- Referential integrity (RI) ensures that data is correct across the database
- Primary keys and foreign keys are properly defined
- Define rules how data is managed in the case of:
  - Insert, update, delete
- Different rules tied to RI
  - Restrict, cascade, default, set null, set default
- Make sure you manage RI properly
- Can use triggers if traditional RI methods are not enough

TRIGGER / CASCADING GRAPH



TRIGGER / CASCADING GRAPH



Create new Player
→ Trigger create Ranking

Create new Match
→ Trigger update Ranking

Update / Delete Match
→ Trigger update Ranking

Delete Player
→ ?????

SUMMARY

- No DBMS can ensure the integrity of its data 100 % reliably all the time
- You must monitor the integrity regularly and every time changes are made
- Both structural and semantic integrities should be checked


LUT
University