

Homework 2

Submit as a Jupyter Notebook by Sunday, December 2017, 23:55

The following relations are used in this homework:

Customers(cid INTEGER PRIMARY KEY, name VARCHAR NOT NULL, dob DATE)

Orders(cid INTEGER NOT NULL, bid INTEGER NOT NULL, price REAL)

Note the absence of keys in the Orders table.

Books(bid INTEGER PRIMARY KEY, title VARCHAR, author VARCHAR)

Basic operators are the 7 basic relational algebra operators (or their SQL counterparts) from slide 68 of Lecture 1.

Exercise 1: (a) Simulate the LEFT and FULL OUTER JOINs between Customers and Orders using the basic SQL operators.

Note that SQL allows arbitrary computed columns, including constants in the SELECT clause:

```
SELECT cid, name, NULL::integer, 1, 'some string'::varchar FROM Customers;
```

The “::type” construct allows to specify the data type of computed columns.

(b) Specify small instances of the relations Customers and Orders above that show the differences between the operators.





(c) **Test your queries on the data from (b) in PostgreSQL!**

Exercise 2: Translate into SQL:



- (a) Select the names of the authors with the highest number of book orders.
- (b) Add number of orders and an average book price to each author name.
- (c) Keep only those authors whose books have generated more than 20'000 in revenues.

Exercise 3: Implement the following queries on the *dvdrental* database from Homework 1 using SQL:

- (a) Find staff members who rented more than 10 film titles in English. 
- (b) What is the maximum number of English titles rented by a single staff member?
- (c) Number of English titles rented by all staff members? 
- (d) Are there French titles that gave more than 100 EUR in rental revenues?
- (e) What are the 3 most popular Italian movies?  

Exercise 4:

(a) Assume that the PK column `oid` in the Orders table, and the following additional table are added to the schema:

```
CREATE TABLE Payments(pid INTEGER, oid INTEGER, payment_day DATE);
```

which indicates when an order has been paid, and the following query:

```
SELECT oid FROM Orders WHERE oid NOT IN (SELECT oid FROM Payments)
```

Would this be a good query for checking unpaid orders? (Consider that the `oid` column in payments could contain nulls). Explain possible problems the query above can lead to.



(b) What could be an intuitive meaning of the following query:

```
SELECT oid FROM Orders
EXCEPT
SELECT oid FROM Orders WHERE oid NOT IN (SELECT oid FROM Payments)
```



(c) With the following small instance

```
INSERT into Customers values (3, 'Bob', NULL);
INSERT into Books values (1, 'Black Swan', 'Taleb');
INSERT into Orders values (10, 3, 1, 25);
INSERT INTO Payments values (101, NULL, '2017-04-05');
```

Explain the output that PostgreSQL produces for the query above on this instance. Is it an expected output for the query explanation from (b)?

Exercise 5. Write down a command for creating a btree-index on customers' first names in the **dvdrental** database and give an example for a range query using that index (index-only scan) providing the respective 'explain' command.

(Note that the command will not work in the **dvdrental** database, which is read-only. You have to create a similar table in your own db to execute such a query).

Exercise 6. Look at the indexes defined in the **dvdrental** database

(**SELECT * FROM pg_indexes WHERE schemaname='public'** or use the `\di` command in psql) and group them according to the purpose these indexes should serve (e.g., making joining tables more efficient, ensuring constraints, filtering rows for specific **WHERE** clauses...). Write down your own query that involves a join and uses one of the indexes. Which join method do you see executed most often?

Exercise 7. The following query is executed by PostgreSQL using the sort merge join algorithm:

```
SELECT * FROM
```

```
(SELECT last_name, first_name, actor_id FROM actor ORDER BY last_name) A
```

```
NATURAL JOIN
```

```
(SELECT last_name, first_name, customer_id FROM customer ORDER BY last_name) C
```

- Analyze the above query in PostgreSQL using the EXPLAIN command and copy the output.
- What changes if you add **DESC** at the end of the first subquery (A)?
Which join method is used now? Explain.
- The sort merge join is very seldom used. Can you guess why? What is the main disadvantage of this algorithm?