Представления, проверки, индексы. Утилита Explain

Определение

- **Представление** (view) — виртуальная (логическая) таблица, представляющая собой **поименованный запрос** (синоним к запросу), который будет подставлен как подзапрос при использовании представления.
- В отличие от обычных таблиц реляционных баз данных, представление не является самостоятельной частью набора данных, хранящегося в базе. Содержимое представления динамически вычисляется на основании данных, находящихся в реальных таблицах. Изменение данных в реальной таблице базы данных немедленно отражается в содержимом всех представлений, построенных на основании этой таблицы.

Создание представления

- **CREATE VIEW** contact_info **AS**

  SELECT cat.name as category, c.first_name, c.middle_name, c.last_name, c.age, ct.protocol, ct.name as contact_type, cv.value FROM contact c, contact_value cv, category cat, contact_type ct WHERE c.id_category=cat.id AND cv.id_contact = c.id AND cv.id_contact_type = ct.id;

- SELECT * FROM **contact_info**;
- SELECT * FROM **contact_info** WHERE age = 20;
- SELECT age, count(*) AS cnt FROM **contact_info** GROUP BY age ORDER BY cnt DESC;

MATERIALIZED представление

- CREATE **MATERIALIZED** VIEW contact_info2 AS

  SELECT cat.name as category, c.first_name, c.middle_name, c.last_name, c.age, ct.protocol, ct.name as contact_type, cv.value FROM contact c, contact_value cv, category cat, contact_type ct WHERE c.id_category=cat.id AND cv.id_contact = c.id AND cv.id_contact_type = ct.id;

- SELECT * FROM **contact_info2**;
- SELECT * FROM **contact_info2** WHERE age = 20;
- SELECT age, count(*) AS cnt FROM **contact_info2** GROUP BY age ORDER BY cnt DESC;

Различие между представлениями

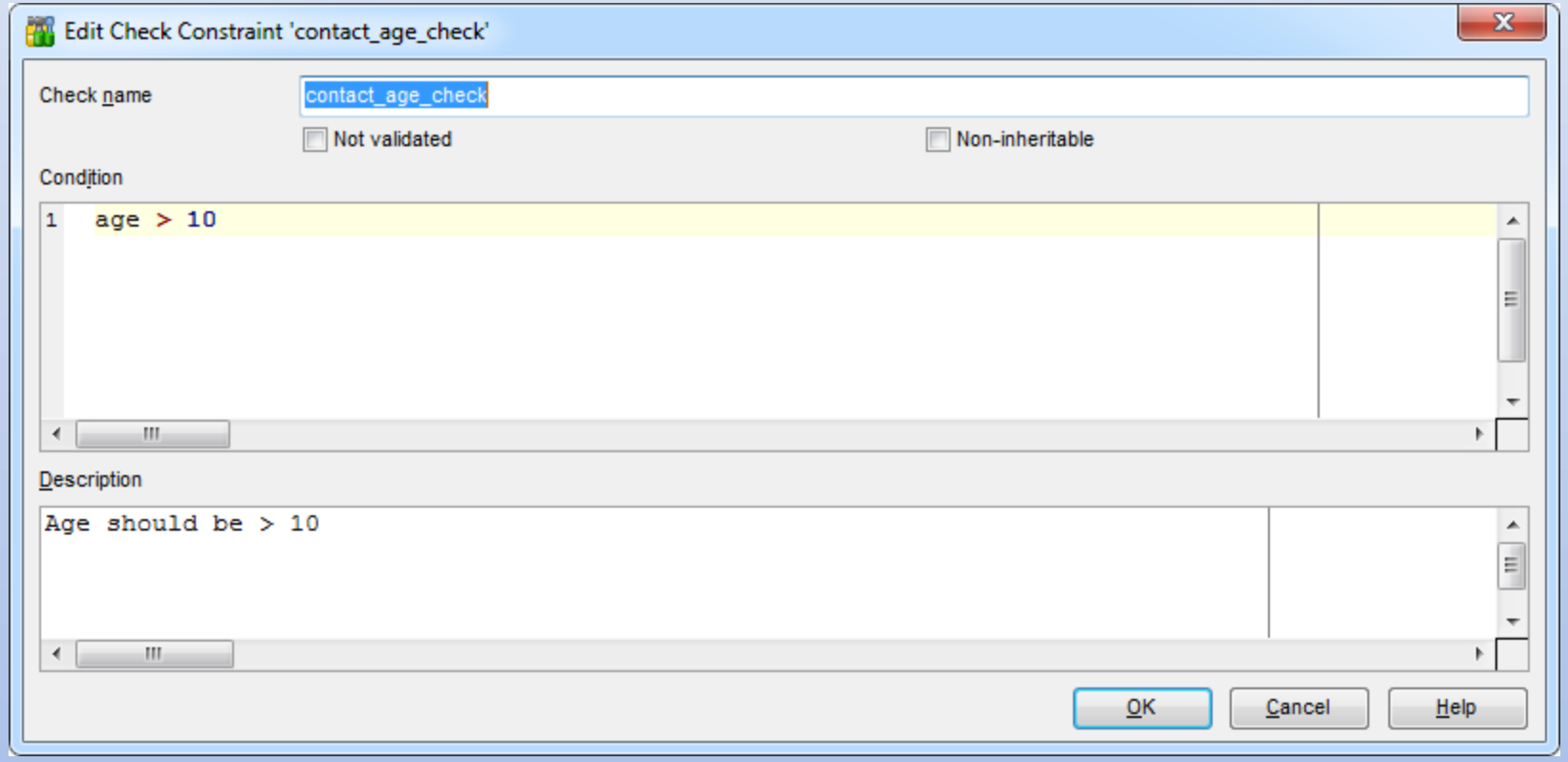
- INSERT INTO **contact_value** VALUES (nextval('contact_value_seq'), 3, 4, 'skype2', now(), NULL, true);

- SELECT * FROM **contact_info**;
- SELECT * FROM **contact_info** WHERE age = 20;
- SELECT age, count(*) AS cnt FROM **contact_info** GROUP BY age ORDER BY cnt DESC;
- SELECT * FROM **contact_info2**;
- SELECT * FROM **contact_info2** WHERE age = 20;
- SELECT age, count(*) AS cnt FROM **contact_info2** GROUP BY age ORDER BY cnt DESC;

- REFRESH MATERIALIZED VIEW contact_info2

- DROP VIEW contact_info
- DROP MATERIALIZED VIEW contact_info2

Проверки



CONSTRAINT contact_age_check CHECK (age > 10),

Индексы

При фильтрации данных из таблицы (Во время выполнения оператора WHERE) необходимо выполнить операцию сравнения параметра запроса с каждым значением поля:

SELECT * FROM contact WHERE age = 20;

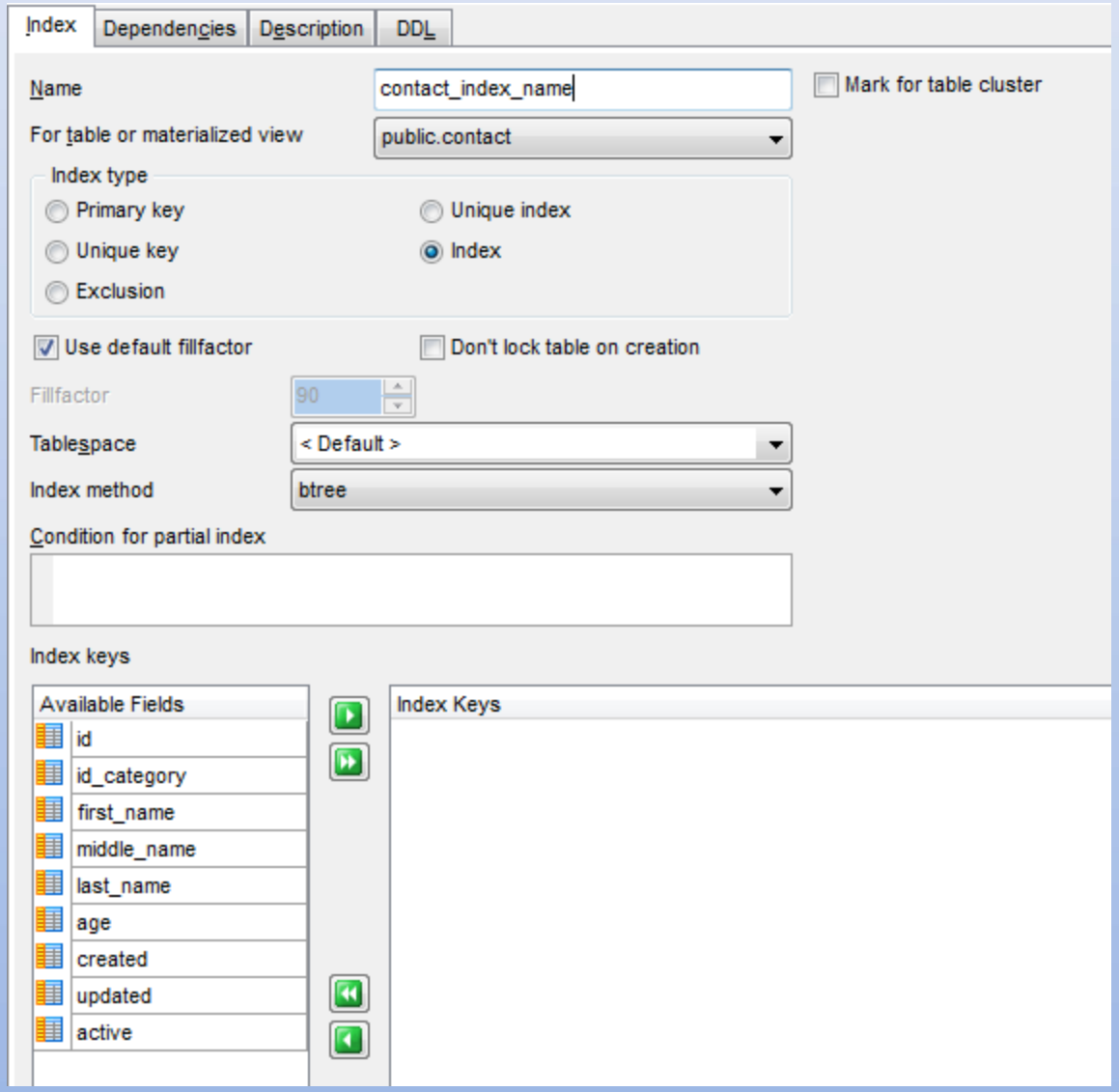
Если данных в таблице много, то время выполнения запроса увеличивается. Для решения данной проблемы используются индексы.

Индекс – это отдельный объект базы данных, который создается по полю (полям) таблицы и хранит копию данных поля (полей) в упорядоченном виде (например в сбалансированном дереве для реализации алгоритма двоичного поиска) и указатели на соответствующие строки таблицы.

Например если создать индекс по полю age таблицы contact, то при выполнении того же самого запроса (SELECT * FROM contact WHERE age = 20) планировщик запросов СУБД может обратиться автоматически к индексу извлечь из него строки, которые удовлетворяют условию выборки и отобразить строки пользователю.

За счет меньшего количества итераций запрос выполнится быстрее!

Индексы



Тестирование индексов

```
CREATE TABLE test (id integer, value text);

INSERT INTO test
  SELECT i, md5(random()::text)
  FROM generate_series(1, 2000000) AS i;

SELECT * FROM test WHERE id = 500;
```

```
1 rows returned (execution time: 156 ms; total time: 172 ms)
```

Создаем индекс по полю id и повторяем запрос:

```
SELECT * FROM test WHERE id = 500;
```

```
1 rows returned (execution time: 0 ms; total time: 0 ms)
```

Индексы

Использование индексов увеличивает производительность запросов к таблице базы данных при этом синтаксис запроса не зависит от наличия индекса, что позволяет создавать индексы уже после проектирования и написания системы;

Планировщик запросов использует индексы не всегда: только в том случае когда использование индексов быстрее чем последовательный перебор. (Ведь используя индекс еще нужно собрать реальные строки по указателям на что тоже тратиться время!)

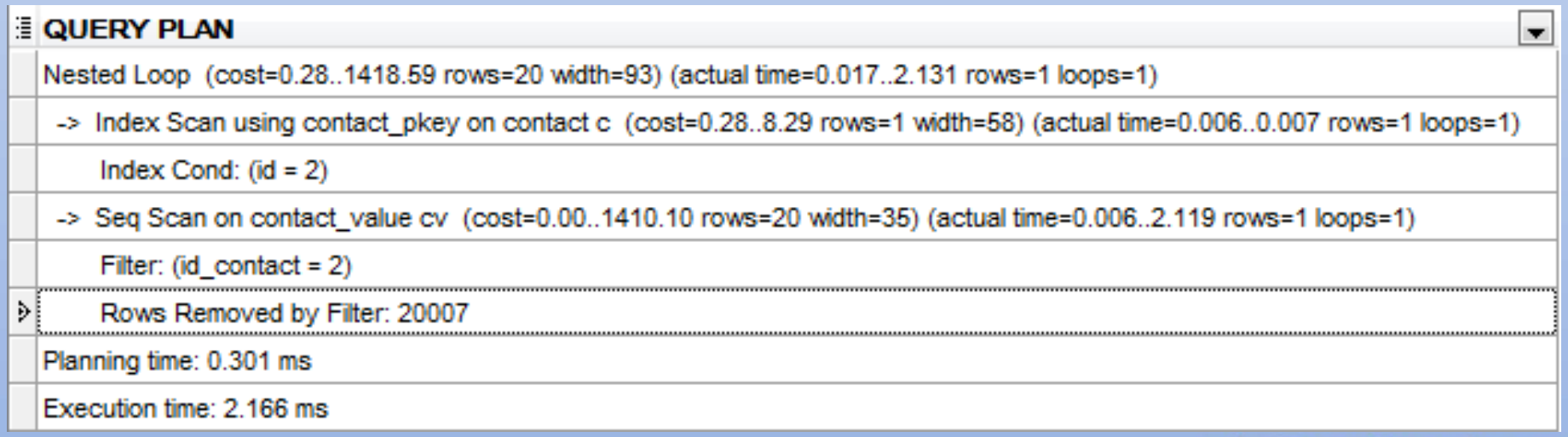
При использовании индексов следует помнить о том, что наличие индексов увеличивает время выполнения INSERT, UPDATE, DELETE – ведь при изменении данных нужно обязательно перестроить индексы по всем полям для которых они созданы!
Кроме того, индексы занимают дополнительный объем памяти, поэтому перед созданием индекса следует убедиться, что планируемый выигрыш в производительности запросов превысит дополнительную затрату ресурсов компьютера на сопровождение индекса.

Для анализа работы планировщика (используются ли индексы или нет) используется утилита EXPLAIN.

EXPLAIN

```
ANALYZE test;
EXPLAIN SELECT * FROM test WHERE id = 500;
EXPLAIN ANALYZE SELECT * FROM test WHERE id = 500;
EXPLAIN ANALYZE SELECT * FROM test WHERE id > 500;
EXPLAIN ANALYZE SELECT * FROM test WHERE id < 500;

ANALYZE contact;
ANALYZE contact_value;
EXPLAIN ANALYZE SELECT * FROM contact WHERE age = 20;
EXPLAIN ANALYZE SELECT * FROM contact c, contact_value cv WHERE c.id = cv.id_contact AND cv.id_contact = 2;
```



QUERY PLAN
Nested Loop (cost=0.28..1418.59 rows=20 width=93) (actual time=0.017..2.131 rows=1 loops=1)
-> Index Scan using contact_pkey on contact c (cost=0.28..8.29 rows=1 width=58) (actual time=0.006..0.007 rows=1 loops=1)
Index Cond: (id = 2)
-> Seq Scan on contact_value cv (cost=0.00..1410.10 rows=20 width=35) (actual time=0.006..2.119 rows=1 loops=1)
Filter: (id_contact = 2)
Rows Removed by Filter: 20007
Planning time: 0.301 ms
Execution time: 2.166 ms

Выводы

1. **Представление** (view) — виртуальная (логическая) таблица, представляющая собой **поименованный запрос** (синоним к запросу), который будет подставлен как подзапрос при использовании представления;
2. Материализированное представление отличается от обычного представления тем, копируются ли данные выборки или нет. Материализированное представление хранит копию данных за счет чего скорость запросов повышается, но при изменении оригинальных данных данные из материализированного представления будут считаться неактуальными. Данные в обычном представлении всегда будут актуальны так как запрос к представлению будет преобразовываться в подзапрос к таблице с реальными данными. Материализированные представления подходят для редко изменяемых данных, или когда точность данные не важна (для статистики), а скорость запроса критична;
3. Проверки (check) позволяют дополнительно валидировать данные перед вставкой в таблицу базы данных;
4. Индекс — объект базы данных, создаваемый с целью повышения скорости поиска данных;
5. Использование индексов увеличивает производительность запросов к таблице базы данных при этом синтаксис запроса не зависит от наличия индекса, что позволяет создавать индексы уже после проектирования и написания системы;
6. При использовании индексов следует помнить о том, что наличие индексов увеличивает время выполнения INSERT, UPDATE, DELETE – ведь при изменении данных нужно обязательно перестроить индексы по всем полям для которых они созданы;
7. Кроме того, индексы занимают дополнительный объем памяти, поэтому перед созданием индекса следует убедиться, что планируемый выигрыш в производительности запросов превысит дополнительную затрату ресурсов компьютера на сопровождение индекса;
8. С помощью утилиты EXPLAIN можно получить план выполнения конкретного запроса и при необходимости его улучшить. Данная утилита присутствует в любой мощной СУБД.