

# Lab\_5\_Nemkov\_KMBO\_05-23

## Task 1

```
CREATE EXTENSION pageinspect;

CREATE TABLE char_table (
    id SERIAL PRIMARY KEY,
    fixed_code CHAR(10)
);

CREATE TABLE varchar_table (
    id SERIAL PRIMARY KEY,
    fixed_code VARCHAR(10)
);

INSERT INTO char_table (fixed_code)
VALUES
    ('Text1'),
    ('LongText2'),
    ('3'),
    ('Text4'),
    ('Text5'),
    ('Text6'),
    ('Text7'),
    ('Text8'),
    ('Text9'),
    ('Text10');

INSERT INTO varchar_table (fixed_code)
VALUES
    ('Text1'),
    ('LongText2'),
    ('3'),
    ('Text4'),
    ('Text5'),
    ('Text6'),
    ('Text7'),
    ('Text8'),
    ('Text9'),
    ('Text10');
```

```
SELECT * FROM heap_page_items(get_raw_page('char_table',0));
```

	lp smallint	lp_off smallint	lp_flags smallint	lp_len smallint	txmin xid	txmax xid	tfield3 integer	tctid tid	tinfomask2 integer	tinfomask integer	thoff smallint	tbits text	toid oid	tdata bytea
1	1	8152	1	39	1241	0	14	(0,1)	2	2050	24	[null]	[null]	[binary dat.
2	2	8112	1	39	1241	0	14	(0,2)	2	2050	24	[null]	[null]	[binary dat.
3	3	8072	1	39	1241	0	14	(0,3)	2	2050	24	[null]	[null]	[binary dat.
4	4	8032	1	39	1241	0	14	(0,4)	2	2050	24	[null]	[null]	[binary dat.
5	5	7992	1	39	1241	0	14	(0,5)	2	2050	24	[null]	[null]	[binary dat.
6	6	7952	1	39	1241	0	14	(0,6)	2	2050	24	[null]	[null]	[binary dat.
7	7	7912	1	39	1241	0	14	(0,7)	2	2050	24	[null]	[null]	[binary dat.
8	8	7872	1	39	1241	0	14	(0,8)	2	2050	24	[null]	[null]	[binary dat.
9	9	7832	1	39	1241	0	14	(0,9)	2	2050	24	[null]	[null]	[binary dat.
10	10	7792	1	39	1241	0	14	(0,10)	2	2050	24	[null]	[null]	[binary dat.

image-302.png

```
SELECT * FROM heap_page_items(get_raw_page('varchar_table',0));
```

	lp smallint	lp_off smallint	lp_flags smallint	lp_len smallint	txmin xid	txmax xid	tfield3 integer	tctid tid	tinfomask2 integer	tinfomask integer	thoff smallint	tbits text	toid oid	tdata bytea
1	1	8152	1	34	1241	0	15	(0,1)	2	2050	24	[null]	[null]	[binary dat.
2	2	8112	1	38	1241	0	15	(0,2)	2	2050	24	[null]	[null]	[binary dat.
3	3	8080	1	30	1241	0	15	(0,3)	2	2050	24	[null]	[null]	[binary dat.
4	4	8040	1	34	1241	0	15	(0,4)	2	2050	24	[null]	[null]	[binary dat.
5	5	8000	1	34	1241	0	15	(0,5)	2	2050	24	[null]	[null]	[binary dat.
6	6	7960	1	34	1241	0	15	(0,6)	2	2050	24	[null]	[null]	[binary dat.
7	7	7920	1	34	1241	0	15	(0,7)	2	2050	24	[null]	[null]	[binary dat.
8	8	7880	1	34	1241	0	15	(0,8)	2	2050	24	[null]	[null]	[binary dat.
9	9	7840	1	34	1241	0	15	(0,9)	2	2050	24	[null]	[null]	[binary dat.
10	10	7800	1	35	1241	0	15	(0,10)	2	2050	24	[null]	[null]	[binary dat.

image-303.png

**Вывод:** CHAR всегда использует фиксированное пространство, тогда как VARCHAR адаптируется под длину данных, что видно по разным значениям `lp_len` для строк разной длины.

	CHAR(n)	VARCHAR(n)
<b>Длина данных</b>	Фиксированная (ровно <code>n</code> байт)	Переменная (фактическая длина + 1-4 байта служебных данных)
<b>Дополнение</b>	Пробелами до указанной длины	Без дополнения
<b>Метаданные</b>	Только заголовок записи	Заголовок + длина значения (1-4 байта)
<b>Пример хранения</b>	'ABC' → 'ABC ' (10 байт)	'ABC' → 'ABC ' (3 байта + 1 байт длины)

**CHAR(n) лучше использовать, если:**

- Данные **всегда фиксированной длины**
- Требуется **предсказуемый размер страницы**

## VARCHAR(n) лучше использовать, если:

- Данные **переменной длины** (имена, описания).
- Важна **экономия места** (не тратится память на пробелы).

## Посмотрим на другие типы данных

```
CREATE TABLE int_table (  
    id SERIAL PRIMARY KEY,  
    value INT  
);  
CREATE TABLE bool_table (  
    id SERIAL PRIMARY KEY,  
    value BOOLEAN  
);  
CREATE TABLE date_table (  
    id SERIAL PRIMARY KEY,  
    value DATE  
);  
CREATE TABLE json_table (  
    id SERIAL PRIMARY KEY,  
    value JSON  
);  
  
INSERT INTO int_table (value) VALUES (1), (-100), (2147483647);  
INSERT INTO bool_table (value) VALUES (TRUE), (FALSE), (NULL);  
INSERT INTO date_table (value) VALUES  
    ('2023-01-01'), ('1970-01-01'), (NULL);  
INSERT INTO json_table (value) VALUES  
    ('{"key": "value"}'), ('[1, 2, 3]'), (NULL);
```

```
SELECT * FROM heap_page_items(get_raw_page('int_table',0));
```

	lp smallint	lp_off smallint	lp_flags smallint	lp_len smallint	txmin xid	txmax xid	tfield3 integer	tctid tid	tinfomask2 integer	tinfomask integer	thoff smallint	tbits text	toid oid	tdata bytea
1	1	8160	1	32	1242	0	32	(0,1)	2	2048	24	[null]	[null]	[binary dat.
2	2	8128	1	32	1242	0	32	(0,2)	2	2048	24	[null]	[null]	[binary dat.
3	3	8096	1	32	1242	0	32	(0,3)	2	2048	24	[null]	[null]	[binary dat.

*image-304.png*

```
SELECT * FROM heap_page_items(get_raw_page('bool_table',0));
```

	lp smallint	lp_off smallint	lp_flags smallint	lp_len smallint	txmin xid	txmax xid	tfield3 integer	tctid tid	tinfomask2 integer	tinfomask integer	thoff smallint	tbits text	toid oid	tdata bytea
1	1	8160	1	29	1242	0	33	(0,1)	2	2048	24	[null]	[null]	[binary d
2	2	8128	1	29	1242	0	33	(0,2)	2	2048	24	[null]	[null]	[binary d
3	3	8096	1	28	1242	0	33	(0,3)	2	2049	24	10000000	[null]	[binary d

image-306.png

```
SELECT * FROM heap_page_items(get_raw_page('date_table',0));
```

	lp smallint	lp_off smallint	lp_flags smallint	lp_len smallint	txmin xid	txmax xid	tfield3 integer	tctid tid	tinfomask2 integer	tinfomask integer	thoff smallint	tbits text	toid oid	tdata bytea
1	1	8160	1	32	1242	0	34	(0,1)	2	2048	24	[null]	[null]	[binary d
2	2	8128	1	32	1242	0	34	(0,2)	2	2048	24	[null]	[null]	[binary d
3	3	8096	1	28	1242	0	34	(0,3)	2	2049	24	10000000	[null]	[binary d

image-307.png

```
SELECT * FROM heap_page_items(get_raw_page('json_table',0));
```

	lp smallint	lp_off smallint	lp_flags smallint	lp_len smallint	txmin xid	txmax xid	tfield3 integer	tctid tid	tinfomask2 integer	tinfomask integer	thoff smallint	tbits text	toid oid	tdata bytea
1	1	8144	1	45	1242	0	35	(0,1)	2	2050	24	[null]	[null]	[binary d
2	2	8104	1	38	1242	0	35	(0,2)	2	2050	24	[null]	[null]	[binary d
3	3	8072	1	28	1242	0	35	(0,3)	2	2049	24	10000000	[null]	[binary d

image-308.png

## Task 2




```
CREATE TABLE accounts (  
    id SERIAL PRIMARY KEY,  
    name VARCHAR(50),  
    balance DECIMAL(10,2)  
);  
  
INSERT INTO accounts (name, balance) VALUES  
    ('Alice', 1000.00),  
    ('Bob', 500.00);
```

Session\_1:

```
BEGIN;  
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  
SELECT * FROM accounts WHERE name = 'Alice';
```

Session\_2:

```
BEGIN;  
UPDATE accounts SET balance = balance + 100 WHERE name = 'Alice';
```

	id [PK] integer 	name character varying (50) 	balance numeric (10,2) 
1	1	Alice	1000.00

*image-309.png*

Стоимость до и после не изменилась, значит мы возможно на уровне `READ COMMITED`

*Попробуем убедиться:*

Session\_1:

```
BEGIN;
SELECT balance FROM accounts WHERE name = 'Alice';
```

Session\_2:

```
UPDATE accounts SET balance = 200 WHERE name = 'Alice';
COMMIT;
```

Session\_3:

```
SELECT balance FROM accounts WHERE name = 'Alice';
COMMIT;
```

Мы видим новые данные, т.е. работаем не со «снимком», а значит уровень – **READ COMMITTED**

Ну и можно проверить уровень, посмотрев на системную переменную `default_transaction_isolation`:

```
SHOW default_transaction_isolation -- Выводит `read committed`
```

## Task 3

### 1. Уровень `READ UNCOMMITTED`

Session\_1:

```
BEGIN;  
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;  
SELECT balance FROM accounts WHERE name = 'Alice'; -- Выводит 200
```

Session\_2:

```
BEGIN;  
UPDATE accounts SET balance = 1100 WHERE name = 'Alice';  
COMMIT;
```

Session\_1:

```
SELECT balance FROM accounts WHERE name = 'Alice'; -- Выводит 1100
```

Изменение видно, хотя наша транзакция не завершена.

*Вывод:* на уровне READ UNCOMMITTED присутствует проблема фантомного чтения.

## 2. Уровень **READ COMMITTED**

Session\_1:

```
BEGIN;  
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  
SELECT balance FROM accounts WHERE name = 'Alice'; -- Выводит 200
```

Session\_2:

```
BEGIN;  
UPDATE accounts SET balance = 1100 WHERE name = 'Alice';  
COMMIT;
```

Session\_1:

```
SELECT balance FROM accounts WHERE name = 'Alice'; -- Выводит 1100
```

Как видно, на уровне READ COMMITTED эта проблема тоже представлена.

---

## 3. Уровень **REPEATABLE READ**

Session\_1:

```
BEGIN;  
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;  
SELECT COUNT(*) FROM accounts; -- Выводит 10
```

Session\_2:

```
BEGIN;  
INSERT INTO accounts VALUES (111, 'Daniil', 111);  
COMMIT;
```

Session\_1:

```
SELECT COUNT(*) FROM accounts; -- Выводит 10
```

Результат не изменился, значит работа со «снимком» на уровне REPEATABLE READ успешно решает проблему фантомного чтения.

Вывод: проблема фантомного чтения решается с уровня изоляции REPEATABLE READ и выше.

---

#### 4. Уровень **SERIALIZABLE**

Session\_1:

```
BEGIN;  
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
UPDATE accounts SET balance = balance + (SELECT SUM(balance) FROM accounts)  
WHERE name = 'Alice';  
COMMIT;
```

Session\_2:

```
BEGIN;  
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
UPDATE accounts SET balance = balance + (SELECT SUM(balance) FROM accounts)  
WHERE name = 'Bob';  
COMMIT;
```

Первая транзакция выполняется без проблем, вторая кидает ошибку:

*"could not serialize access"*

Следовательно, на уровне SERIALIZABLE аномалия сериализации невозможна.