**НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ ПО БАЗАМ ДАННЫХ**

студента 3 курса, группы 22205

Рыленко Валентина Григорьевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Рылов Сергей Александрович

Новосибирск 2025

**СОДЕРЖАНИЕ**

[ЗАДАНИЕ 3](#_heading=h.eh53vinq2rgv)

[ОПИСАНИЕ РАБОТЫ](#_heading=h.p2ztp8b58agv) 4-6

[ЗАКЛЮЧЕНИЕ](#_heading=h.dv6axg4gx5ff) 6

Приложение 1. Запросы создания таблиц.7-9

Приложение 2. Триггеры10-11

Приложение 3. Запросы12-21

# **ЗАДАНИЕ**

Для выбранного проекта студент разрабатывает структуру базы данных и реализует приложение в архитектуре клиент-сервер, выполняющее операции внесения данных в базу данных, редактирование данных и запросы, указанные в проекте. Клиентская часть реализуется на языке программирования высокого уровня.

# **ОПИСАНИЕ РАБОТЫ**

Схема таблиц была сделана на сайте SQLDBM: [app.sqldbm.com/PostgreSQL/DatabaseExplorer/p330003/](https://app.sqldbm.com/PostgreSQL/DatabaseExplorer/p330003/).

Реализация проекта доступна в репозитории <https://github.com/rylenko/db2sem>.

Созданы следующие таблицы:

* Клубы (clubs). Содержит идентификатор клуба, его название и время создания в базе данных.
* Спортсмены (sportsmen). Содержит идентификатор спортсмена, его имя, дату рождения, рост, вес, идентификатор клуба, к которому спортсмен принадлежит, и время создания спортсмена в базе данных. Построена связь многие к одному с таблицей клубов.
* Виды спорта (sports). Содержит идентификатор спорта, его название и время создания в базе данных.
* Спортсмены и виды спорта, которыми они занимаются (sportsman\_sports). Является связью многие ко многим между спортсменами и видами спорта. Содержит идентификатор связи, идентификатор спортсмена, идентификатор спорта и разряд спортсмена в данном виде спорта.
* Тренеры (trainers). Содержит идентификатор тренера, его имя и время создания в базе данных.
* Тренера спортсменов, занимающихся определённым видом спорта (sportsman\_sport\_trainers). Является связью многие ко многим между связью спортсмена со спортом и тренерами. Содержит идентификатор связи, идентификатор связи спортсмена и вида спорта, которая тоже является связью многие ко многим, и идентификатор тренера. Два поля образуют первичный ключ.
* Организации (organizers). Содержит идентификатор организации, её название, расположение и время создания в базе данных.
* Типы сооружений (place\_types). Содержит идентификатор типа сооружений, его название и название таблицы атрибутов типа сооружения.
* Сооружения (places). Содержит идентификатор сооружения, его название, расположение, идентификатор типа атрибута и время создания в базе данных. Построена связь многие к одному с таблицей типов сооружений.
* Атрибуты манежей (arena\_attributes). Содержит идентификатор сооружения, для которого созданы атрибуты. Идентификатор сооружения является первичным ключом. Также содержит количество рефери и длина беговой дорожки.
* Атрибуты стадионов (stadium\_attributes). Содержит идентификатор сооружения, для которого созданы атрибуты. Идентификатор сооружения является первичным ключом. Также содержит ширину и длину стадиона в сантиметрах, максимальное количество зрителей, булево значение, определяющее, является ли стадион уличным, и покрытие стадиона.
* Атрибуты спортивных залов (gym\_attributes). Содержит идентификатор сооружения, количество тренеров, количество гантель и налачие бани.
* Атрибуты кортов (court\_attributes). Содержит идентификатор места, ширину и длину корта в сантиметрах и флаг, является ли корт открытым.
* Соревнования (tournaments). Содержит идентификатор соревнования, идентификатор сооружения, где проходит соревнование, время начала, идентификатор организатора и время создания в базе данных. Построена связь многие к одному с таблицей сооружений, также построена связь многие к одному с таблицей организаторов.
* Виды спорта в соревновании (tournament\_sports). Содержит идентификатор связи, идентификатор соревнования и идентификатор спорта. Таблица является связью многие ко многим между таблицами соревнований и видов спорта.
* Участия в соревнованиях (participations). Содержит идентификатор связи, идентификатор вида спорта в соревновании, идентификатор спортсмена, место в соревновании и результат участия в виде текста. Идентификатор спорта в соревновании образует связь многие к одному с таблицей спортов в соревновании. Идентификатор спортсмена образует связь многие к одному с таблицей спортсменов.

Определены следующие ограничения целостности:

* Среди всех наборов атрибутов идентификатор сооружения должен быть уникальным. Так невозможно присвоить более одного набора атрибутов одному сооружению. Более того, необходима проверка, что набор атрибутов, где указан идентификатор сооружения, соотносится с типом сооружения. То есть, например, если у сооружения тип стадиона, то есть возможность использовать только таблицу с атрибутами стадиона. Этот вопрос решён так: разработчик создаёт новую таблицу атрибутов. После добавляет в таблицу типов сооружения новый тип с любым именем и с названием таблицы атрибутов, которую создал ранее. После добавления типа сооружения для таблицы атрибутов автоматически создастся триггер, вызывающийся перед каждой вставкой и обновлением. Триггер будет проверять, что идентификатор сооружения указывает на сооружение, тип которого имеет таблицу атрибутов равную текущей. Уникальность среди всех таблиц атрибутов достигается с помощью первичного ключа, установленного на поле идентификатора места в таблицах атрибутов.

Согласно заданию реализованы 13 запросов. Запросы находятся в приложении 3.

**ЗАКЛЮЧЕНИЕ**

В первом этапеспроектирована модель, определены сущности и связи между ними. Определены ограничения целостности, написаны SQL скрипты создания таблиц БД.

Во втором этапе реализованы запросы к базе данных согласно заданию.

В третьем этапе согласно ограничениям целостности из первого этапа реализованы триггеры.

# **Приложение 1.** Запросы создания таблиц

1. CREATE TABLE IF NOT EXISTS place\_types (
2. id BIGSERIAL PRIMARY KEY,
3. name TEXT NOT NULL UNIQUE,
4. attributes\_table\_name TEXT NOT NULL UNIQUE
5. );
6. CREATE TABLE IF NOT EXISTS places (
7. id BIGSERIAL PRIMARY KEY,
8. name VARCHAR(255) NOT NULL,
9. location TEXT NOT NULL,
10. type\_id BIGINT NOT NULL REFERENCES place\_types(id) ON DELETE CASCADE,
11. created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
12. );
13. CREATE TABLE IF NOT EXISTS arena\_attributes (
14. place\_id BIGINT PRIMARY KEY REFERENCES places(id) ON DELETE CASCADE,
15. referees\_count SMALLINT NOT NULL CHECK (referees\_count >= 0),
16. treadmill\_length\_cm BIGINT NOT NULL CHECK (treadmill\_length\_cm > 0)
17. );
18. CREATE TABLE IF NOT EXISTS court\_attributes (
19. place\_id BIGINT PRIMARY KEY REFERENCES places(id) ON DELETE CASCADE,
20. width\_cm BIGINT NOT NULL CHECK (width\_cm > 0),
21. length\_cm BIGINT NOT NULL CHECK (length\_cm > 0),
22. is\_outdoor BOOLEAN NOT NULL
23. );
24. CREATE TABLE IF NOT EXISTS gym\_attributes (
25. place\_id BIGINT PRIMARY KEY REFERENCES places(id) ON DELETE CASCADE,
26. trainers\_count SMALLINT NOT NULL CHECK (trainers\_count >= 0),
27. dumbbells\_count SMALLINT NOT NULL CHECK (dumbbells\_count >= 0),
28. has\_bathhouse BOOLEAN NOT NULL
29. );
30. CREATE TABLE IF NOT EXISTS stadium\_attributes (
31. place\_id BIGINT PRIMARY KEY REFERENCES places(id) ON DELETE CASCADE,
32. width\_cm BIGINT NOT NULL CHECK (width\_cm > 0),
33. length\_cm BIGINT NOT NULL CHECK (length\_cm > 0),
34. max\_spectators SMALLINT NOT NULL CHECK (max\_spectators >= 0),
35. is\_outdoor BOOLEAN NOT NULL,
36. coating VARCHAR(255) NOT NULL
37. );
38. CREATE TABLE IF NOT EXISTS clubs (
39. id BIGSERIAL PRIMARY KEY,
40. name VARCHAR(255) NOT NULL,
41. created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
42. );
43. CREATE TABLE IF NOT EXISTS organizers (
44. id BIGSERIAL PRIMARY KEY,
45. name VARCHAR(255) NOT NULL,
46. location TEXT,
47. created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
48. );
49. CREATE TABLE IF NOT EXISTS sports (
50. id BIGSERIAL PRIMARY KEY,
51. name VARCHAR(255) NOT NULL UNIQUE,
52. created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
53. );
54. CREATE TABLE IF NOT EXISTS tournaments (
55. id BIGSERIAL PRIMARY KEY,
56. place\_id BIGINT NOT NULL REFERENCES places(id) ON DELETE CASCADE,
57. start\_at TIMESTAMPTZ NOT NULL,
58. organizer\_id BIGINT NOT NULL REFERENCES organizers(id) ON DELETE CASCADE,
59. created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
60. );
61. CREATE TABLE IF NOT EXISTS tournament\_sports (
62. id BIGSERIAL PRIMARY KEY,
63. tournament\_id BIGINT NOT NULL REFERENCES tournaments(id) ON DELETE CASCADE,
64. sport\_id BIGINT NOT NULL REFERENCES sports(id) ON DELETE CASCADE,
65. UNIQUE (tournament\_id, sport\_id)
66. );
67. CREATE TABLE IF NOT EXISTS sportsmen (
68. id BIGSERIAL PRIMARY KEY,
69. name VARCHAR(255) NOT NULL,
70. birth\_date DATE NOT NULL,
71. height\_cm SMALLINT NOT NULL CHECK (height\_cm > 0),
72. weight\_kg NUMERIC(5, 2) NOT NULL CHECK (weight\_kg > 0),
73. club\_id BIGINT NOT NULL REFERENCES clubs(id) ON DELETE CASCADE,
74. created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
75. );
76. CREATE TABLE IF NOT EXISTS participations (
77. id BIGSERIAL PRIMARY KEY,
78. tournament\_sport\_id BIGINT NOT NULL REFERENCES tournament\_sports(id) ON DELETE CASCADE,
79. sportsman\_id BIGINT NOT NULL REFERENCES sportsmen(id) ON DELETE CASCADE,
80. rank SMALLINT NOT NULL CHECK (rank >= 0),
81. results TEXT,
82. UNIQUE (tournament\_sport\_id, sportsman\_id)
83. );
84. CREATE TABLE IF NOT EXISTS sportsman\_sports (
85. id BIGSERIAL PRIMARY KEY,
86. sportsman\_id BIGINT NOT NULL REFERENCES sportsmen(id) ON DELETE CASCADE,
87. sport\_id BIGINT NOT NULL REFERENCES sports(id) ON DELETE CASCADE,
88. rank SMALLINT CHECK (rank IS NULL OR rank >= 0),
89. UNIQUE (sportsman\_id, sport\_id)
90. );
91. CREATE TABLE IF NOT EXISTS trainers (
92. id BIGSERIAL PRIMARY KEY,
93. name VARCHAR(255) NOT NULL,
94. created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
95. );
96. CREATE TABLE IF NOT EXISTS sportsman\_sport\_trainers (
97. id BIGSERIAL PRIMARY KEY,
98. sportsman\_sport\_id BIGINT NOT NULL REFERENCES sportsman\_sports(id) ON DELETE CASCADE,
99. trainer\_id BIGINT NOT NULL REFERENCES trainers(id) ON DELETE CASCADE,
100. UNIQUE (sportsman\_sport\_id, trainer\_id)
101. );

# Приложение 2. Триггеры

-- Проверяет, что сооружение с идентификатором place\_id находится в верной таблице атрибутов.

CREATE OR REPLACE FUNCTION validate\_place\_id\_in\_correct\_attributes\_table()

RETURNS TRIGGER AS $$

DECLARE

required\_table\_name TEXT;

place\_type RECORD;

place\_ids\_count INT;

BEGIN

SELECT attributes\_table\_name

INTO required\_table\_name

FROM place\_types

WHERE id = (SELECT type\_id FROM places WHERE id = NEW.place\_id);

IF required\_table\_name IS NULL THEN

RAISE EXCEPTION 'required table name not found for place id %', NEW.place\_id;

END IF;

IF TG\_TABLE\_NAME <> required\_table\_name THEN

RAISE EXCEPTION 'attributes of place with ID % must be in % table', NEW.place\_id, required\_table\_name;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- Триггер, который добавляет триггер валидации place\_id таблицы атрибутов для новой записи типа сооружений.

CREATE OR REPLACE FUNCTION add\_validate\_place\_id\_in\_correct\_attributes\_table()

RETURNS TRIGGER AS $$

DECLARE

query TEXT;

BEGIN

query := format(

'CREATE OR REPLACE TRIGGER validate\_%I\_trigger

BEFORE INSERT OR UPDATE ON %I

FOR EACH ROW EXECUTE FUNCTION validate\_place\_id\_in\_correct\_attributes\_table();',

NEW.attributes\_table\_name,

NEW.attributes\_table\_name

);

EXECUTE query;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER add\_validate\_place\_id\_in\_correct\_attributes\_table\_trigger

BEFORE INSERT OR UPDATE ON place\_types

FOR EACH ROW EXECUTE FUNCTION add\_validate\_place\_id\_in\_correct\_attributes\_table();

# Приложение 3. Запросы

-- Query #1.1

--

-- Получить перечень спортивных сооружений указанного типа в целом или

-- удовлетворяющих заданным характеристикам (например, стадионы, вмещающие не менее

-- указанного числа зрителей).

--

-- name: GetArenaPlaces :many

SELECT

p.name,

p.location

FROM places p

JOIN arena\_attributes aa ON aa.place\_id = p.id

WHERE

(

aa.referees\_count >= sqlc.narg('referees\_count')

OR sqlc.narg('referees\_count') IS NULL

)

AND (

aa.treadmill\_length\_cm >= sqlc.narg('treadmill\_length\_cm')

OR sqlc.narg('treadmill\_length\_cm') IS NULL

);

-- Query #1.2

--

-- Получить перечень спортивных сооружений указанного типа в целом или

-- удовлетворяющих заданным характеристикам (например, стадионы, вмещающие не менее

-- указанного числа зрителей).

--

-- name: GetStadiumPlaces :many

SELECT

p.name,

p.location

FROM places p

JOIN stadium\_attributes sa ON sa.place\_id = p.id

WHERE

(

sa.width\_cm >= sqlc.narg('width\_cm')

OR sqlc.narg('width\_cm') IS NULL

)

AND (

sa.length\_cm >= sqlc.narg('length\_cm')

OR sqlc.narg('length\_cm') IS NULL

)

AND (

sa.max\_spectators >= sqlc.narg('max\_spectators')

OR sqlc.narg('max\_spectators') IS NULL

)

AND (

sa.is\_outdoor = sqlc.narg('is\_outdoor')

OR sqlc.narg('is\_outdoor') IS NULL

)

AND (

sa.coating = sqlc.narg('coating')

OR sqlc.narg('coating') IS NULL

);

-- Query #1.3

--

-- Получить перечень спортивных сооружений указанного типа в целом или

-- удовлетворяющих заданным характеристикам (например, стадионы, вмещающие не менее

-- указанного числа зрителей).

--

-- name: GetCourtPlaces :many

SELECT

p.name,

p.location

FROM places p

JOIN court\_attributes ca ON ca.place\_id = p.id

WHERE

(

ca.width\_cm >= sqlc.narg('width\_cm')

OR sqlc.narg('width\_cm') IS NULL

)

AND (

ca.length\_cm >= sqlc.narg('length\_cm')

OR sqlc.narg('length\_cm') IS NULL

)

AND (

ca.is\_outdoor = sqlc.narg('is\_outdoor')

OR sqlc.narg('is\_outdoor') IS NULL

);

-- Query #1.4

--

-- Получить перечень спортивных сооружений указанного типа в целом или

-- удовлетворяющих заданным характеристикам (например, стадионы, вмещающие не менее

-- указанного числа зрителей).

--

-- name: GetGymPlaces :many

SELECT

p.name,

p.location

FROM places p

JOIN gym\_attributes ga ON ga.place\_id = p.id

WHERE

(

ga.trainers\_count >= sqlc.narg('trainers\_count')

OR sqlc.narg('trainers\_count') IS NULL

)

AND (

ga.dumbbells\_count >= sqlc.narg('dumbbells\_count')

OR sqlc.narg('dumbbells\_count') IS NULL

)

AND (

ga.has\_bathhouse = sqlc.narg('has\_bathhouse')

OR sqlc.narg('has\_bathhouse') IS NULL

);

-- Query #2

--

-- Получить список спортсменов, занимающихся указанным видом спорта в целом либо не

-- ниже определенного разряда.

--

-- name: GetSportsmenBySportID :many

SELECT

sm.name,

sm.birth\_date,

sm.height\_cm,

sm.weight\_kg

FROM sportsmen sm

JOIN sportsman\_sports ss ON ss.sportsman\_id = sm.id

WHERE

ss.sport\_id = @sport\_id

AND (

ss.rank >= sqlc.narg('rank')

OR sqlc.narg('rank') IS NULL

);

-- Query #3

--

-- Получить список спортсменов, тренирующихся у некого тренера в целом либо не ниже

-- определенного разряда.

--

-- name: GetSportsmenByTrainerID :many

SELECT

sm.name,

sm.birth\_date,

sm.height\_cm,

sm.weight\_kg

FROM sportsmen sm

JOIN sportsman\_sports ss ON ss.sportsman\_id = sm.id

JOIN sportsman\_sport\_trainers sst ON sst.sportsman\_sport\_id = ss.id

WHERE

sst.trainer\_id = @trainer\_id

AND (

ss.rank >= sqlc.narg('rank')

OR sqlc.narg('rank') IS NULL

);

-- Query #4

--

-- Получить список спортсменов, занимающихся более чем одним видом спорта с указанием

-- этих видов спорта.

--

-- name: GetSportsmenInvolvedInSeveralSports :many

SELECT

sm.id,

sm.name,

sm.birth\_date,

sm.height\_cm,

sm.weight\_kg,

ARRAY\_AGG(s.name)::TEXT[] AS sport\_names

FROM sportsmen sm

JOIN sportsman\_sports sms ON sms.sportsman\_id = sm.id

JOIN sports s ON s.id = sms.sport\_id

GROUP BY

sm.id,

sm.name,

sm.birth\_date,

sm.height\_cm,

sm.weight\_kg

HAVING COUNT(sms.id) > 1

ORDER BY sm.id;

-- Query #5

--

-- Получить список тренеров указанного спортсмена.

--

-- name: GetTrainersBySportsmanID :many

SELECT t.name

FROM trainers t

JOIN sportsman\_sport\_trainers sst ON sst.trainer\_id = t.id

JOIN sportsman\_sports ss ON ss.id = sst.sportsman\_sport\_id

WHERE ss.sportsman\_id = $1;

-- Query #6

--

-- Получить перечень соревнований, проведенных в течение заданного периода времени в

-- целом либо указанным организатором.

--

-- name: GetTournamentsForPeriod :many

SELECT

p.name,

o.name,

t.start\_at

FROM tournaments t

JOIN places p ON p.id = t.place\_id

JOIN organizers o ON o.id = t.organizer\_id

WHERE

t.start\_at BETWEEN @start\_at AND @end\_at

AND (

t.organizer\_id = sqlc.narg('organizer\_id')

OR sqlc.narg('organizer\_id') IS NULL

);

-- Query #7

--

-- Получить список призеров указанного соревнования.

--

-- name: GetSportsmenByTournamentID :many

SELECT

sm.name,

sm.birth\_date,

sm.height\_cm,

sm.weight\_kg

FROM sportsmen sm

JOIN participations p ON p.sportsman\_id = sm.id

JOIN tournament\_sports ts ON ts.id = p.tournament\_sport\_id

WHERE

ts.tournament\_id = $1

AND p.rank <= 3

ORDER BY p.rank;

-- Query #8

--

-- Получить перечень соревнований, проведенных в указанном спортивном сооружении в

-- целом либо по определенному виду спорта.

--

-- name: GetTournamentsByPlaceID :many

SELECT

p.name,

o.name,

t.start\_at

FROM tournaments t

JOIN places p ON p.id = t.place\_id

JOIN organizers o ON o.id = t.organizer\_id

JOIN tournament\_sports ts ON ts.tournament\_id = t.id

WHERE

t.place\_id = $1

AND (

ts.sport\_id = sqlc.narg('sport\_id')

OR sqlc.narg('sport\_id') IS NULL

);

-- Query #9

--

-- Получить перечень спортивных клубов и число спортсменов этих клубов, участвовавших в

-- спортивных соревнованиях в течение заданного интервала времени.

--

-- name: GetClubActiveSportsmenCountsForPeriod :many

SELECT

c.name,

COUNT(s.id)

FROM clubs c

LEFT JOIN sportsmen s ON s.club\_id = c.id

LEFT JOIN participations p ON p.sportsman\_id = s.id

LEFT JOIN tournament\_sports ts ON ts.id = p.tournament\_sport\_id

LEFT JOIN tournaments t ON t.id = ts.tournament\_id

WHERE t.start\_at BETWEEN @start\_at AND @end\_at

GROUP BY

c.id,

c.name;

-- Query #10

--

-- Получить список тренеров по определенному виду спорта.

--

-- name: GetTrainersBySportID :many

SELECT t.name

FROM trainers t

JOIN sportsman\_sport\_trainers sst ON sst.trainer\_id = t.id

JOIN sportsman\_sports ss ON ss.id = sst.sportsman\_sport\_id

WHERE ss.sport\_id = $1;

-- Query: #11

--

-- Получить список спортсменов, не участвовавших ни в каких соревнованиях в течение

-- определенного периода времени.

--

-- name: GetInactiveSportsmenForPeriod :many

SELECT

sm.name,

sm.birth\_date,

sm.height\_cm,

sm.weight\_kg

FROM sportsmen sm

WHERE NOT EXISTS (

SELECT 1

FROM participations p

JOIN tournament\_sports ts ON ts.id = p.tournament\_sport\_id

JOIN tournaments t ON t.id = ts.tournament\_id

WHERE

t.start\_at BETWEEN @start\_at AND @end\_at

AND p.sportsman\_id = sm.id

);

-- Query: #12

--

-- Получить список организаторов соревнований и число проведенных ими соревнований в

-- течение определенного периода времени.

--

-- name: GetOrganizerTournamentCountsForPeriod :many

SELECT

o.name,

o.location,

COUNT(t.id)

FROM organizers o

LEFT JOIN tournaments t ON t.organizer\_id = o.id

GROUP BY

o.id,

o.name,

o.location;

-- Query: #13

--

-- Получить перечень спортивных сооружений и даты проведения на них соревнований в

-- течение определенного периода времени.

--

-- name: GetPlaceTournamentDatesForPeriod :many

SELECT

p.name,

p.location,

ARRAY\_AGG(t.start\_at)::TIMESTAMPTZ[] as dates

FROM places p

LEFT JOIN tournaments t ON t.place\_id = p.id

WHERE t.start\_at BETWEEN @start\_at AND @end\_at

GROUP BY

p.id,

p.name,

p.location;

-- Query: #14 (custom)

--

-- Создаёт манеж и задаёт для него аттрибуты.

--

-- name: InsertArena :exec

WITH

place\_type AS (

SELECT id FROM place\_types WHERE attributes\_table\_name = 'arena\_attributes'

),

place AS (

INSERT INTO places (name, location, type\_id)

VALUES (@name, @location, (SELECT id FROM place\_type))

RETURNING id

)

INSERT INTO arena\_attributes (place\_id, referees\_count, treadmill\_length\_cm)

VALUES ((SELECT id FROM place), @referees\_count, @treadmill\_length\_cm);

-- Query: #15 (custom)

--

-- Создаёт стадион и задаёт для него аттрибуты.

--

-- name: InsertStadium :exec

WITH

place\_type AS (

SELECT id FROM place\_types WHERE attributes\_table\_name = 'stadium\_attributes'

),

place AS (

INSERT INTO places (name, location, type\_id)

VALUES (@name, @location, place\_type.id)

RETURNING id

)

INSERT INTO stadium\_attributes (place\_id, width\_cm, length\_cm, max\_spectators, is\_outdoor, coating)

VALUES ((SELECT id FROM place), @width\_cm, @length\_cm, @max\_spectators, @is\_outdoor, @coating);

-- Query: #16 (custom)

--

-- Создаёт корт и задаёт для него аттрибуты.

--

-- name: InsertCourt :exec

WITH

place\_type AS (

SELECT id FROM place\_types WHERE attributes\_table\_name = 'court\_attributes'

),

place AS (

INSERT INTO places (name, location, type\_id)

VALUES (@name, @location, place\_type.id)

RETURNING id

)

INSERT INTO court\_attributes (place\_id, width\_cm, length\_cm, is\_outdoor)

VALUES ((SELECT id FROM place), @width\_cm, @length\_cm, @is\_outdoor);

-- Query: #17 (custom)

--

-- Создаёт зал и задаёт для него аттрибуты.

--

-- name: InsertGym :exec

WITH

place\_type AS (

SELECT id FROM place\_types WHERE attributes\_table\_name = 'gym\_attributes'

),

place AS (

INSERT INTO places (name, location, type\_id)

VALUES (@name, @location, place\_type.id)

RETURNING id

)

INSERT INTO gym\_attributes (place\_id, trainers\_count, dumbbells\_count, has\_bathhouse)

VALUES ((SELECT id FROM place), @trainers\_count, @dumbbells\_count, @has\_bathhouse);

-- Query: #18 (custom)

--

-- Получает спортсмена по идентификатору.

--

-- name: GetSportsmanByID :one

SELECT

sm.id,

sm.name,

sm.birth\_date,

sm.height\_cm,

sm.weight\_kg,

(ARRAY\_AGG(s.name) FILTER (WHERE s.name IS NOT NULL))::TEXT[] AS sport\_names

FROM sportsmen sm

LEFT JOIN sportsman\_sports sms ON sms.sportsman\_id = sm.id

LEFT JOIN sports s ON s.id = sms.sport\_id

WHERE sm.id = $1

GROUP BY

sm.id,

sm.name,

sm.birth\_date,

sm.height\_cm,

sm.weight\_kg;

-- Query: #19 (custom)

--

-- Получает названия всех спортов.

--

-- name: GetSportNames :many

SELECT name

FROM sports;

-- Query: #20 (custom)

--

-- Удаляет спортсмена по ID.

--

-- name: DeleteSportsmanByID :exec

DELETE FROM sportsmen

WHERE id = $1;

-- Query: #21 (custom)

--

-- Получает всех спортсменов.

--

-- name: GetSportsmen :many

SELECT

sm.id,

sm.name,

sm.birth\_date,

sm.height\_cm,

sm.weight\_kg,

(ARRAY\_AGG(s.name) FILTER (WHERE s.name IS NOT NULL))::TEXT[] AS sport\_names

FROM sportsmen sm

LEFT JOIN sportsman\_sports sms ON sms.sportsman\_id = sm.id

LEFT JOIN sports s ON s.id = sms.sport\_id

GROUP BY

sm.id,

sm.name,

sm.birth\_date,

sm.height\_cm,

sm.weight\_kg

ORDER BY sm.id;

-- Query: #22 (custom)

--

-- Обновляет спортсмена по идентификатору.

--

-- name: UpdateSportsmanByID :exec

WITH old\_sportsman\_sports AS (

DELETE FROM sportsman\_sports

WHERE sportsman\_id = $1

),

new\_sportsman\_sports AS (

INSERT INTO sportsman\_sports (sportsman\_id, sport\_id)

SELECT $1, s.id

FROM UNNEST(@sport\_names::TEXT[]) as sport\_name

JOIN sports s ON s.name = sport\_name

ON CONFLICT (sportsman\_id, sport\_id) DO NOTHING

)

UPDATE sportsmen AS sm

SET

name = $2,

birth\_date = $3,

height\_cm = $4,

weight\_kg = $5

WHERE sm.id = $1;