

Task 1:

Insert:

Table: match

```
Choose table:
  press 1 - Match...   press 2 - Stadium...   press 3 - StartDate... press 4 - Team
1
Do you want to insert rows manually or randomly?
Press M if manually, R if randomly: M
What number of row do you want to add? - 1
Enter values following this sequence: Match_id, opp_score, own_score
Match_id: 3232
opp_score: 4
own_score: 4
Successfully inserted
```

717	1	1
51	5	5
3232	4	4

```
Enter values following this sequence: Match_id, opp_score, own_score
Match_id: 321
opp_score: 2
own_score: 2
Failed inserting record into table (psycopg2.errors.UniqueViolation) ПОМИЛКА: повторювані значення ключа порушують обмеження унікальності "Match_pkey"
DETAIL: Ключ (match_id)=(321) вже існує.

[SQL: INSERT INTO match (match_id, opp_score, own_score) VALUES (%(match_id)s, %(opp_score)s, %(own_score)s)]
[parameters: {'match_id': '321', 'opp_score': '2', 'own_score': '2'}]
(Background on this error at: http://sqlalche.me/e/13/gkpi)
Do you want to continue? Press Y if yes, N if no: |
```

Table: stadium

```
  press 1 - Match...   press 2 - Stadium...   press 3 - StartDate... press 4 - Team
2
Do you want to insert rows manually or randomly?
Press M if manually, R if randomly: M
What number of row do you want to add? - 1
Enter values following this sequence: Stadium_id, Name, City
Stadium_id: 232
Name: name
City: city
Successfully inserted
```

3	get	got
54	ar	ft

```
  press 1 - Match...   press 2 - Stadium...   press 3 - StartDate... press 4 - Team
4
Do you want to insert rows manually or randomly?
Press M if manually, R if randomly: M
What number of row do you want to add? - 1
Enter values following this sequence: Team_id, Name, Opponent, Coach, Stadium_id
Enter values following this sequence: Team_id, Match_id, Start_date(yyyy-mm-dd)
Team_id: 7
Match_id: 1
Start_date: 2020-10-10
Failed inserting record into table (psycopg2.errors.UniqueViolation) ПОМИЛКА: повторювані значення ключа порушують обмеження унікальності "PK_StartDate"
DETAIL: Ключ (team_id, match_id)=(7, 1) вже існує.

[SQL: INSERT INTO startdate (match_id, team_id, start_date) VALUES (%(match_id)s, %(team_id)s, %(start_date)s)]
[parameters: {'match_id': '1', 'team_id': '7', 'start_date': '2020-10-10'}]
(Background on this error at: http://sqlalche.me/e/13/gkpi)
Do you want to continue? Press Y if yes, N if no: |
```

Table: team

```
Choose table:
press 1 - Match...   press 2 - Stadium...   press 3 - StartDate... press 4 - Team
4
Do you want to insert rows manually or randomly?
Press M if manually, R if randomly: M
What number of row do you want to add? - 1
Enter values following this sequence: Team_id, Name, Opponent, Coach, Stadium_id
Team_id: 132
Name: nam
Opponent: opan
Coach: coa
Stadium_id: 3
Successfully inserted
```

```
2100477      5be8b5154f708795053bf6ac7745d45f e34a6d3c96519136e0ec8bdc484d74f7 55214af4bd0e969190a03119fe6f7ed3 2
2100478      fhf          fgf          fdf          3
3423         n          op          co          3
```

```
Enter values following this sequence: Team_id, Name, Opponent, Coach, Stadium_id
Team_id: 11232
Name: fg
Opponent: fg
Coach: fg
Stadium_id: 434535
Failed inserting record into table (psycopg2.errors.ForeignKeyViolation) ПОМИЛКА: insert або update в таблиці "team" порушує обмеження зовнішнього ключа "Team"
DETAIL: Ключ (stadium_id)=(434535) не присутній в таблиці "stadium".

[SQL: INSERT INTO team (team_id, name, opponent, coach, stadium_id) VALUES (%(team_id)s, %(name)s, %(opponent)s, %(coach)s, %(stadium_id)s)]
[parameters: {'team_id': '11232', 'name': 'fg', 'opponent': 'fg', 'coach': 'fg', 'stadium_id': '434535'}]
(Background on this error at: http://sqlalche.me/e/13/qkcy)
Do you want to continue? Press Y if yes, N if no: |
```

Update:

Table: match

```
Choose table:
press 1 - Match...   press 2 - Stadium...   press 3 - StartDate... press 4 - Team
1
What number of rows do you want to update? - 1
Enter values following this sequence opp_score, own_score, Match_id:
opp_score: 5
own_score: 5
Match_id: 51
Successfully updated
Do you want to continue? Press Y if yes, N if no: |
```

835	321	500
51	672	850
577	159	298

717	1	1
51	5	5
3232	4	4

Table: stadium

Enter values following this sequence Name, City, Stadium_id:

Name: *gr*

City: *ft*

Stadium_id: *54*

Successfully updated

Do you want to continue? Press Y if yes, N if no: |

12	222f	dfd
54	fe	wr
7	XTFRKJ	QOMFVD

3	get	got
54	gr	ft
232	name	city

Table: startdate

Choose table:

press 1 - Match... press 2 - Stadium... press 3 - StartDate... press 4 - Team

3

What number of rows do you want to update? - *1*

Enter values following this sequence Start_date(yyyy-mm-dd), Team_id, Match_id:

Start_date: *2012-09-09*

Team_id: *2094100*

Match_id: *212*

Successfully updated

2094100	212	2000-01-10
7	1	2020-09-09

7	1	2020-09-09
2094100	212	2012-09-09

Table: team

press 1 - Match... press 2 - Stadium... press 3 - StartDate... press 4 - Team

4

What number of rows do you want to update? - *1*

Enter values following this sequence Name, Opponent, Coach, Stadium_id, Team_id:

Name: *fhf*

Opponent: *fgf*

Coach: *fdf*

Stadium_id: *3*

Team_id: *2100478*

Successfully updated

Do you want to continue? Press Y if yes, N if no:

2100478	6427ab105780510fc393ff66f9685300	b41c45397aaaf7040526b5d99a827996	6a098a260b1c953535c2c6ebdc659820	1
132	nam	opon	coa	3

132	nam	opon	coa	3
2100478	fhf	fgf	fdf	3

Delete:

Table: match

717	1	1
321	2	3
51	5	5

```
What number of row do you want to delete? - 1
Enter value that marks Match_id:51
Failed deleting record into table (psycopg2.errors.ForeignKeyViolation) ПОМИЛКА:
DETAIL:  На ключ (match_id)=(51) все ще є посилання в таблиці "startdate".

[SQL: DELETE FROM match WHERE match.match_id = %(match_id)s]
[parameters: {'match_id': 51}]
(Background on this error at: http://sqlalche.me/e/13/gkpi)
Do you want to continue? Press Y if yes, N if no:
```

```
press 1 - Match... press 2 - Stadium... press 3 - StartDate... press 4 - Team
1
What number of row do you want to delete? - 1
Enter value that marks Match_id:460
Match_id: 460

1 Record deleted
```

418	976	296
460	221	619
486	88	228

418	976	296
486	88	228

Table: stadium

121	name	city
3	get	got
54	gr	ft

```
Enter value that marks Stadium_id:121
Successfully deleted
Do you want to continue? Press Y if yes, N if no:
What number of row do you want to delete? - 1
Enter value that marks Stadium_id:54
Failed deleting record into table (psycopg2.errors.ForeignKeyViolation) ПОМИЛКА:
DETAIL:  На ключ (stadium_id)=(54) все ще є посилання в таблиці "team".

[SQL: DELETE FROM stadium WHERE stadium.stadium_id = %(stadium_id)s]
[parameters: {'stadium_id': 54}]
(Background on this error at: http://sqlalche.me/e/13/gkpi)
```

Table: startdate

```
Enter values following this sequence team_id, match_id:
team_id: 5
match_id: 5

1 Record deleted
```

3	4	2000-02-01
5	5	2020-09-25
7	3	2019-12-12

3	4	2000-02-01
7	3	2019-12-12

```
What number of row do you want to delete? - 1
Enter values following this sequence team_id, match_id:
team_id: 1
match_id: 453
Failed deleting record into table No row was found for one()
Failed deleting record into table Class 'builtins.str' is not mapped
```

Table: team

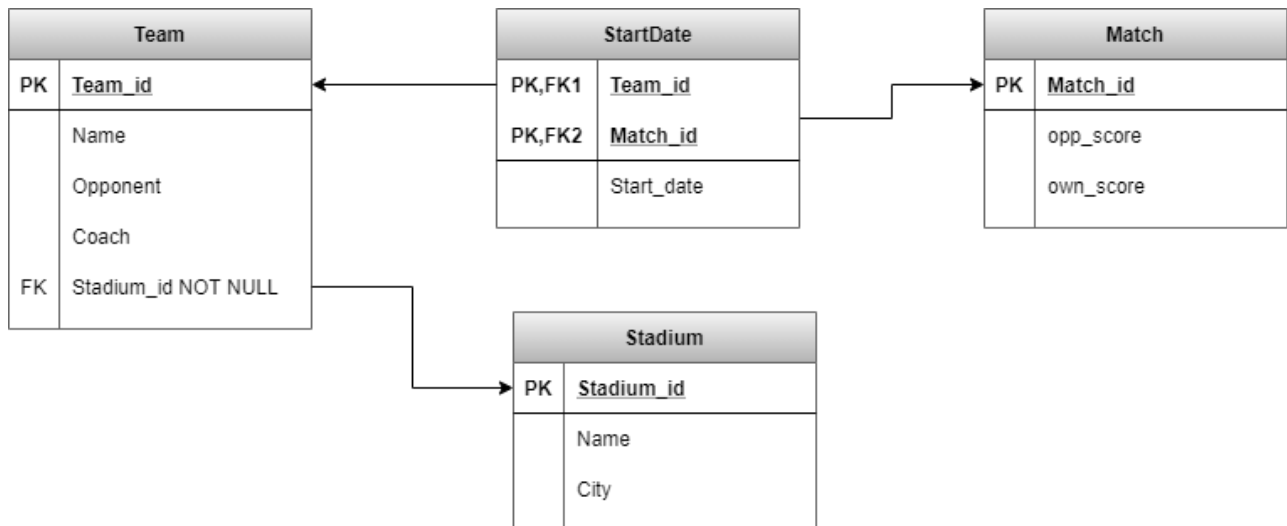
```
What number of row do you want to delete? - 1
Enter value that mark Team_id:132
Successfully deleted
Do you want to continue? Press Y if yes, N if no: |
```

2100477	5be8b5154f708795053bf6ac7745d45f	e34a6d3c96519136e0ec8bdc484d74f7	55214af4bd0e969190a03119fe6f7ed3	2
132	nam	opon	coa	3
2100478	fhf	fgf	fdf	3

2100477	5be8b5154f708795053bf6ac7745d45f	e34a6d3c96519136e0ec8bdc484d74f7	55214af4bd0e969190a03119fe6f7ed3	2
2100478	fhf	fgf	fdf	3

```
What number of row do you want to delete? - 1
Enter value that mark Team_id:223
Failed deleting record into table No row was found for one()
Failed deleting record into table Class 'builtins.str' is not mapped
```

Relational database schema:



Base classes in SQLAlchemy ORM:

```
class Match(Model.Base):
    __tablename__ = 'match'
    match_id = Column(Integer, primary_key=True)
    opp_score = Column(Integer)
    own_score = Column(Integer)
    team_s = relationship("Startdate", back_populates="match_r")

    def __init__(self, match_id, opp_score, own_score):
        self.match_id = match_id
        self.opp_score = opp_score
        self.own_score = own_score
```

```
class Team(Model.Base):
    __tablename__ = 'team'
    team_id = Column(Integer, primary_key=True)
    name = Column(String(32))
    opponent = Column(String(32))
    coach = Column(String(32))
    stadium_id = Column(Integer, ForeignKey('stadium.stadium_id'))
    stadium = relationship("Stadium")
    match_s = relationship("Startdate", back_populates="team_r")

    def __init__(self, team_id, name, opponent, coach, stadium_id):
```

```
class Startdate(Model.Base):
    __tablename__ = 'startdate'
    match_id = Column(Integer, ForeignKey('match.match_id'), primary_key=True)
    team_id = Column(Integer, ForeignKey('team.team_id'), primary_key=True)
    start_date = Column(Date)
    match_r = relationship("Match", back_populates="team_s")
    team_r = relationship("Team", back_populates="match_s")

    def __init__(self, match_id, team_id, start_date):
        self.team_id = team_id
        self.match_id = match_id
        self.start_date = start_date
```

```

class Stadium(Model.Base):
    __tablename__ = 'stadium'
    stadium_id = Column(Integer, primary_key=True)
    name = Column(String(32))
    city = Column(String(32))

    def __init__(self, stadium_id, name, city):
        self.stadium_id = stadium_id
        self.name = name
        self.city = city

```

To implement a database using SQLAlchemy ORM, you need to create a unique class for each table with the corresponding ones characteristics (representation of columns, table names, data types, keys and connections between them). The Match and Stadium tables used the Many to One relationship type, as many matches are in the same stadium and one stadium is in many matches. One-way communication type used.

In the process of normalization, the Many to Many relationship created an additional table that references the primary keys of these two tables, so the standard the structure is not used. Instead, we use two Many to One bindings that belong to the same Startdate table with a composite primary key. These tables use a two-way link type.

Task 2:

GIN

GIN stands for generalized inverse index. It does not index the values themselves, but individual elements; each element refers to the values in which it occurs. An ordered set of table rows that contain values with this element is bound to each element. Items are never removed from the GIN. This solution significantly simplifies the algorithms, providing parallel work with the index of several processes.

If the TID (tuple identifier) is small, it is placed on the same page as the item (and is called the placement list). Otherwise, a B-tree is used for an efficient data structure. The disadvantage of this method is that inserting or updating data is slow due to the large number of tokens that need to be indexed. The advantage is good compactness. The same token is always stored once. Also, the TID is stored in the index in an orderly manner, which allows you to use compression: each subsequent

TID list is stored as a difference from the previous one - usually a small number that requires far fewer bits than the full TID.

	Query Editor	Query History
1	INSERT INTO team (name, opponent, coach, stadium_id)	
2	SELECT nam, opp, coach, stadium_id FROM	
3	(SELECT md5((random()*1)::text) as nam,	
4	md5((random()*2)::text) as opp,	
5	md5((random()*3)::text) as coach,	
6	stadium_id	
7	FROM stadium tablesample BERNOULLI(100)	
8	ORDER BY random()) k, generate_series(1, 100000) LIMIT 900000	

Data Output	Explain	Messages	Notifications
INSERT 0 900000			
Query returned successfully in 1 min 57 secs.			

Comand_Match/postgres@PostgreSQL 12

	Query Editor	Query History
1	select count(*) from team	

Data Output	Explain	Messages	Notifications
<div>count bigint</div>			
1	2900068		

	Query Editor	Query History
1	Explain select * from team	

Data Output	Explain	Messages	Notifications
<div>QUERY PLAN text</div>			
1	Seq Scan on team (cost=0.00..27243.68 rows=1000068 width=104)		


```

1 CREATE EXTENSION IF NOT EXISTS pg_trgm;
2 CREATE INDEX gin_idx ON team USING gin(name gin_trgm_ops);

```

Data Output Explain Messages Notifications

ПОВІДОМЛЕННЯ: розширення "pg_trgm" вже існує, пропускаємо
CREATE INDEX

Query returned successfully in 29 secs 222 msec.

As you can see the editor chooses the best option for each case. Under the conditions shown below, sequential search is used, as most of the values fall under this search.

Query Editor Query History

```

1 explain analyze SELECT * FROM team where name ilike '%b%';

```

Data Output Explain Messages Notifications

QUERY PLAN		
	text	
1	Seq Scan on team (cost=0.00..29743.85 rows=904995 width=104) (actual time=0.103..3713.988 rows=905277 loops=1)	
2	Filter: ((name)::text ~~* '%b%':text)	
3	Rows Removed by Filter: 94791	
4	Planning Time: 0.316 ms	
5	Execution Time: 3759.748 ms	



Comand_Match/postgres@PostgreSQL 12

Query Editor Query History

```

1 --explain analyze
2 SELECT count(*) FROM team where name ilike '%ab%';

```

Data Output Explain Messages Notifications

count		
	bigint	
1	94738	

In this case, a parallel sequential search is performed, because the GIN index may not always be compatible with this type of search. Since indexing is not done by value, but by individual parts of this element.

Comand_Match/postgres@PostgreSQL 12																							
Query Editor Query History																							
<pre> 1 explain analyze 2 SELECT count(*) FROM team where name ilike '%ab%'; </pre>																							
Data Output Explain Messages Notifications																							
<table> <thead> <tr> <th></th><th>QUERY PLAN</th></tr> </thead> <tbody> <tr> <td>1</td><td>Finalize Aggregate (cost=23550.60..23550.61 rows=1 width=8) (actual time=2144.589..2154.290 rows=1 loops=1)</td></tr> <tr> <td>2</td><td>-> Gather (cost=23550.39..23550.60 rows=2 width=8) (actual time=2144.177..2154.280 rows=3 loops=1)</td></tr> <tr> <td>3</td><td>Workers Planned: 2</td></tr> <tr> <td>4</td><td>Workers Launched: 2</td></tr> <tr> <td>5</td><td>-> Partial Aggregate (cost=22550.39..22550.40 rows=1 width=8) (actual time=2058.118..2058.119 rows=1 loops=3)</td></tr> <tr> <td>6</td><td>-> Parallel Seq Scan on team (cost=0.00..22451.69 rows=39480 width=0) (actual time=234.005..2051.982 rows=31579 lo...</td></tr> <tr> <td>7</td><td>Filter: ((name)::text ~~* '%ab%':text)</td></tr> <tr> <td>8</td><td>Rows Removed by Filter: 301777</td></tr> <tr> <td>9</td><td>Planning Time: 0.358 ms</td></tr> <tr> <td>10</td><td>Execution Time: 2154.329 ms</td></tr> </tbody> </table>			QUERY PLAN	1	Finalize Aggregate (cost=23550.60..23550.61 rows=1 width=8) (actual time=2144.589..2154.290 rows=1 loops=1)	2	-> Gather (cost=23550.39..23550.60 rows=2 width=8) (actual time=2144.177..2154.280 rows=3 loops=1)	3	Workers Planned: 2	4	Workers Launched: 2	5	-> Partial Aggregate (cost=22550.39..22550.40 rows=1 width=8) (actual time=2058.118..2058.119 rows=1 loops=3)	6	-> Parallel Seq Scan on team (cost=0.00..22451.69 rows=39480 width=0) (actual time=234.005..2051.982 rows=31579 lo...	7	Filter: ((name)::text ~~* '%ab%':text)	8	Rows Removed by Filter: 301777	9	Planning Time: 0.358 ms	10	Execution Time: 2154.329 ms
	QUERY PLAN																						
1	Finalize Aggregate (cost=23550.60..23550.61 rows=1 width=8) (actual time=2144.589..2154.290 rows=1 loops=1)																						
2	-> Gather (cost=23550.39..23550.60 rows=2 width=8) (actual time=2144.177..2154.280 rows=3 loops=1)																						
3	Workers Planned: 2																						
4	Workers Launched: 2																						
5	-> Partial Aggregate (cost=22550.39..22550.40 rows=1 width=8) (actual time=2058.118..2058.119 rows=1 loops=3)																						
6	-> Parallel Seq Scan on team (cost=0.00..22451.69 rows=39480 width=0) (actual time=234.005..2051.982 rows=31579 lo...																						
7	Filter: ((name)::text ~~* '%ab%':text)																						
8	Rows Removed by Filter: 301777																						
9	Planning Time: 0.358 ms																						
10	Execution Time: 2154.329 ms																						

Next, the index gin search is used. It works as follows. Initially, the search query selects tokens (search keys): 'f1%' ('a%'). Then, using the B-tree or the key page, the finished TID lists are searched. Then, for each TID list found, a special function is executed that determines the values that fall under the search query. And bitmap scan is used because the result is returned as a bit cards. As a result, index search shows better performance.

Comand_Match/postgres@PostgreSQL 12																							
Query Editor Query History																							
<pre> 1 explain analyze 2 select * from team where name ilike 'f1%' or name ilike 'a%' </pre>																							
Data Output Explain Messages Notifications																							
<table> <thead> <tr> <th></th><th>QUERY PLAN</th></tr> </thead> <tbody> <tr> <td>1</td><td>Bitmap Heap Scan on team (cost=1246.85..53174.48 rows=128084 width=107) (actual time=44.931..1313.646 rows=123078 loops=1)</td></tr> <tr> <td>2</td><td>Recheck Cond: (((name)::text ~~* 'f1%':text) OR ((name)::text ~~* 'a%':text))</td></tr> <tr> <td>3</td><td>Heap Blocks: exact=34484</td></tr> <tr> <td>4</td><td>-> BitmapOr (cost=1246.85..1246.85 rows=128375 width=0) (actual time=34.083..34.084 rows=0 loops=1)</td></tr> <tr> <td>5</td><td>-> Bitmap Index Scan on gin_idx (cost=0.00..84.20 rows=6960 width=0) (actual time=2.582..2.582 rows=7692 loops=1)</td></tr> <tr> <td>6</td><td>Index Cond: ((name)::text ~~* 'f1%':text)</td></tr> <tr> <td>7</td><td>-> Bitmap Index Scan on gin_idx (cost=0.00..1098.61 rows=121415 width=0) (actual time=31.499..31.499 rows=115386 loops=1)</td></tr> <tr> <td>8</td><td>Index Cond: ((name)::text ~~* 'a%':text)</td></tr> <tr> <td>9</td><td>Planning Time: 0.637 ms</td></tr> <tr> <td>10</td><td>Execution Time: 1321.811 ms</td></tr> </tbody> </table>			QUERY PLAN	1	Bitmap Heap Scan on team (cost=1246.85..53174.48 rows=128084 width=107) (actual time=44.931..1313.646 rows=123078 loops=1)	2	Recheck Cond: (((name)::text ~~* 'f1%':text) OR ((name)::text ~~* 'a%':text))	3	Heap Blocks: exact=34484	4	-> BitmapOr (cost=1246.85..1246.85 rows=128375 width=0) (actual time=34.083..34.084 rows=0 loops=1)	5	-> Bitmap Index Scan on gin_idx (cost=0.00..84.20 rows=6960 width=0) (actual time=2.582..2.582 rows=7692 loops=1)	6	Index Cond: ((name)::text ~~* 'f1%':text)	7	-> Bitmap Index Scan on gin_idx (cost=0.00..1098.61 rows=121415 width=0) (actual time=31.499..31.499 rows=115386 loops=1)	8	Index Cond: ((name)::text ~~* 'a%':text)	9	Planning Time: 0.637 ms	10	Execution Time: 1321.811 ms
	QUERY PLAN																						
1	Bitmap Heap Scan on team (cost=1246.85..53174.48 rows=128084 width=107) (actual time=44.931..1313.646 rows=123078 loops=1)																						
2	Recheck Cond: (((name)::text ~~* 'f1%':text) OR ((name)::text ~~* 'a%':text))																						
3	Heap Blocks: exact=34484																						
4	-> BitmapOr (cost=1246.85..1246.85 rows=128375 width=0) (actual time=34.083..34.084 rows=0 loops=1)																						
5	-> Bitmap Index Scan on gin_idx (cost=0.00..84.20 rows=6960 width=0) (actual time=2.582..2.582 rows=7692 loops=1)																						
6	Index Cond: ((name)::text ~~* 'f1%':text)																						
7	-> Bitmap Index Scan on gin_idx (cost=0.00..1098.61 rows=121415 width=0) (actual time=31.499..31.499 rows=115386 loops=1)																						
8	Index Cond: ((name)::text ~~* 'a%':text)																						
9	Planning Time: 0.637 ms																						
10	Execution Time: 1321.811 ms																						

Query Editor Query History

```

1 explain analyze
2 select * from team where name ilike 'f1%' --or name ilike '%a3%'

```

Data Output Explain Messages Notifications

	QUERY PLAN	
	text	
1	Bitmap Heap Scan on team (cost=85.94..19161.27 rows=6960 width=107) (actual time=1.693..26.780 rows=7692 loops=1)	
2	Recheck Cond: ((name)::text ~~* 'f1%':text)	
3	Heap Blocks: exact=1725	
4	-> Bitmap Index Scan on gin_idx (cost=0.00..84.20 rows=6960 width=0) (actual time=1.453..1.453 rows=7692 loops=1)	
5	Index Cond: ((name)::text ~~* 'f1%':text)	
6	Planning Time: 0.251 ms	
7	Execution Time: 27.246 ms	

Query Editor Query History

```

1 explain analyze
2 select count(*) from team where name ilike 'f1%' --or name ilike '%a3%'

```

Data Output Explain Messages Notifications

	QUERY PLAN	
	text	
1	Aggregate (cost=19178.67..19178.68 rows=1 width=8) (actual time=29.623..29.624 rows=1 loops=1)	
2	-> Bitmap Heap Scan on team (cost=85.94..19161.27 rows=6960 width=0) (actual time=1.798..28.992 rows=7692 loops=1)	
3	Recheck Cond: ((name)::text ~~* 'f1%':text)	
4	Heap Blocks: exact=1725	
5	-> Bitmap Index Scan on gin_idx (cost=0.00..84.20 rows=6960 width=0) (actual time=1.590..1.590 rows=7692 loops=1)	
6	Index Cond: ((name)::text ~~* 'f1%':text)	
7	Planning Time: 0.395 ms	
8	Execution Time: 29.816 ms	

Query Editor		Query History
1	explain analyze	
2	select count(*) from team where name ilike 'f1%' group by stadium_id--or name ilike 'a%'	
Data Output		Explain Messages Notifications
	QUERY PLAN	
	text	
1	HashAggregate (cost=19196.07..19196.26 rows=19 width=12) (actual time=68.648..68.649 rows=1 loops=1)	
2	Group Key: stadium_id	
3	-> Bitmap Heap Scan on team (cost=85.94..19161.27 rows=6960 width=4) (actual time=4.542..64.389 rows=7692 loops=1)	
4	Recheck Cond: ((name)::text ~* 'f1%':text)	
5	Heap Blocks: exact=1725	
6	-> Bitmap Index Scan on gin_idx (cost=0.00..84.20 rows=6960 width=0) (actual time=4.000..4.000 rows=7692 loops=1)	
7	Index Cond: ((name)::text ~* 'f1%':text)	
8	Planning Time: 0.446 ms	
9	Execution Time: 68.857 ms	

Comand_Match/postgres@PostgreSQL 12		
Query Editor		Query History
1	explain analyze	
2	select * from team where name ilike 'f1%' or name ilike 'a%' order by team_id desc	
Data Output		Explain Messages Notifications
	QUERY PLAN	
	text	
1	Sort (cost=71484.81..71805.02 rows=128084 width=107) (actual time=1412.538..1442.567 rows=123078 loops=1)	
2	Sort Key: team_id DESC	
3	Sort Method: external merge Disk: 14208kB	
4	-> Bitmap Heap Scan on team (cost=1246.85..53174.48 rows=128084 width=107) (actual time=31.308..1286.200 rows=123078 loops=1)	
5	Recheck Cond: (((name)::text ~* 'f1%':text) OR ((name)::text ~* 'a%':text))	
6	Heap Blocks: exact=34484	
7	-> BitmapOr (cost=1246.85..1246.85 rows=128375 width=0) (actual time=25.342..25.343 rows=0 loops=1)	
8	-> Bitmap Index Scan on gin_idx (cost=0.00..84.20 rows=6960 width=0) (actual time=2.359..2.359 rows=7692 loops=1)	
9	Index Cond: ((name)::text ~* 'f1%':text)	
10	-> Bitmap Index Scan on gin_idx (cost=0.00..1098.61 rows=121415 width=0) (actual time=22.980..22.980 rows=115386 loops=1)	
11	Index Cond: ((name)::text ~* 'a%':text)	
12	Planning Time: 0.789 ms	
13	Execution Time: 1557.235 ms	

BRIN

The idea of BRIN indexing is to create a range of blocks or groups of pages that are adjacent to each other and information about them is stored in the index. That is, the higher priority is to avoid looking at unnecessary lines. This method works well for data that is already virtually sorted, as the physical location will correlate with the value of the columns. Otherwise, the advantage of the blocks will be neglected. Therefore, this type of index is convenient to apply to numerical data or dates. The advantage is the relatively small size and minimal resources to maintain operation.

The algorithm is as follows:

The map of sites is consistently viewed. With the help of pointers index lines with summary information for each section are determined. If the plot does not exactly contain the desired value, it is skipped; if you can contain (or if summary information is missing) - all plot pages are added to the bitmap. This bitmap is used in further search. And while this algorithm may sacrifice efficiency, it will benefit from maintaining high data capacity.

 Comand_Match/postgres@PostgreSQL 12


Query Editor Query History

1 `create index brin_indx on startdate using brin(start_date)`

Data Output Explain Messages Notifications

CREATE INDEX


Query returned successfully in 99 msec.

 Comand_Match/postgres@PostgreSQL 12

Query Editor Query History

1 `--explain analyze`
2 `select count(*) from startdate`

Data Output Explain Messages Notifications

 Comand_Match/postgres@PostgreSQL 12

Query Editor Query History

1 `--explain analyze`
2 `select * from startdate where start_date between '2008-01-01' and '2008-01-13'`

Data Output Explain Messages Notifications

	team_id [PK] integer	match_id [PK] integer	start_date date
1	2000482	3	2008-01-01
2	2000481	3	2008-01-02
3	2000483	3	2008-01-03
4	2000484	3	2008-01-04
5	2000485	3	2008-01-05
6	2000486	3	2008-01-06
7	2000487	3	2008-01-07
8	2000488	3	2008-01-08
9	2000489	3	2008-01-09
10	2000490	3	2008-01-10
11	2000491	3	2008-01-11
12	2000492	3	2008-01-12
13	2000493	3	2008-01-13

Because the query includes > 90% of all rows, a sequential search is performed automatically.

Command_Match/postgres@PostgreSQL 12

Query Editor

Query History

1

explain analyze

2

select * from startdate where start_date < '2008-01-01'

Data Output

Explain

Messages

Notifications

▲

QUERY PLAN

text

🔒

1

Seq Scan on startdate (cost=0.00..1794.50 rows=100190 width=12) (actual time=0.039..22.514 rows=100007 loops=1)

2

Filter: (start_date < '2008-01-01'::date)

3

Rows Removed by Filter: 24

4

Planning Time: 0.259 ms

5

Execution Time: 26.545 ms

The default query uses sequential search.

Comand_Match/postgres@PostgreSQL 12

Query Editor

Query History

1

explain analyze

2

select * from startdate where start_date between '2008-01-01' and '2008-01-13'

Data Output

Explain

Messages

Notifications

QUERY PLAN

text

1

Seq Scan on startdate (cost=0.00..2042.46 rows=1 width=12) (actual time=14.315..14.318 rows=13 loops=1)

2

Filter: ((start_date >= '2008-01-01'::date) AND (start_date <= '2008-01-13'::date))

3

Rows Removed by Filter: 100018

4

Planning Time: 1.870 ms

5

Execution Time: 14.345 ms

The same query, but with the BRIN index significantly faster.



```
1 explain analyze
2 select * from startdate where start_date between '2008-01-01' and '2008-01-13'
```

QUERY PLAN		
	text	
1	Bitmap Heap Scan on startdate (cost=12.03..854.63 rows=1 width=12) (actual time=3.682..3.687 rows=13 loops=1)	
2	Recheck Cond: ((start_date >= '2008-01-01'::date) AND (start_date <= '2008-01-13'::date))	
3	Rows Removed by Index Recheck: 28979	
4	Heap Blocks: lossy=158	
5	-> Bitmap Index Scan on brin_indx (cost=0.00..12.03 rows=20040 width=0) (actual time=0.023..0.023 rows=2560 loops=1)	
6	Index Cond: ((start_date >= '2008-01-01'::date) AND (start_date <= '2008-01-13'::date))	
7	Planning Time: 0.172 ms	
8	Execution Time: 3.723 ms	

Comand_Match/postgres@PostgreSQL 12

Query Editor

Query History

1

explain analyze

2

select count(*) from startdate where start_date between '2008-01-01' and '2008-01-13' group by match_id

Data Output

Explain

Messages

Notifications

QUERY PLAN

text

1

GroupAggregate (cost=854.64..854.66 rows=1 width=12) (actual time=6.840..6.841 rows=1 loops=1)

2

Group Key: match_id

3

-> Sort (cost=854.64..854.65 rows=1 width=4) (actual time=6.826..6.828 rows=13 loops=1)

4

Sort Key: match_id

5

Sort Method: quicksort Memory: 25kB

6

-> Bitmap Heap Scan on startdate (cost=12.03..854.63 rows=1 width=4) (actual time=6.798..6.808 rows=13 loops=1)

7

Recheck Cond: ((start_date >= '2008-01-01'::date) AND (start_date <= '2008-01-13'::date))

8

Rows Removed by Index Recheck: 28979

9

Heap Blocks: lossy=158

10

-> Bitmap Index Scan on brin_indx (cost=0.00..12.03 rows=20040 width=0) (actual time=0.034..0.034 rows=2560 loops=1)

11

Index Cond: ((start_date >= '2008-01-01'::date) AND (start_date <= '2008-01-13'::date))

12

Planning Time: 0.180 ms

13

Execution Time: 6.914 ms

Comand_Match/postgres@PostgreSQL 12

Query Editor

Query History

1

explain analyze

2

select count(*) from startdate where start_date between '2008-01-01' and '2008-01-13'

Data Output

Explain

Messages

Notifications

QUERY PLAN

text

1

Aggregate (cost=854.63..854.64 rows=1 width=8) (actual time=8.331..8.332 rows=1 loops=1)

2

-> Bitmap Heap Scan on startdate (cost=12.03..854.63 rows=1 width=0) (actual time=8.317..8.324 rows=13 loops=1)

3

Recheck Cond: ((start_date >= '2008-01-01'::date) AND (start_date <= '2008-01-13'::date))

4

Rows Removed by Index Recheck: 28979

5

Heap Blocks: lossy=158

6

-> Bitmap Index Scan on brin_indx (cost=0.00..12.03 rows=20040 width=0) (actual time=0.038..0.038 rows=2560 loops=1)

7


Index Cond: ((start_date >= '2008-01-01'::date) AND (start_date <= '2008-01-13'::date))

8

Planning Time: 0.157 ms

9

Execution Time: 8.395 ms


Comand_Match/postgres@PostgreSQL 12

Query Editor
Query History

```

1  explain analyze
2  select * from startdate where start_date between '2008-01-01' and '2008-01-13' order by match_id, team_id asc

```

Data Output
Explain
Messages
Notifications

	QUERY PLAN	
	text	
1	Sort (cost=854.64..854.65 rows=1 width=12) (actual time=7.947..7.950 rows=13 loops=1)	
2	Sort Key: match_id, team_id	
3	Sort Method: quicksort Memory: 25kB	
4	-> Bitmap Heap Scan on startdate (cost=12.03..854.63 rows=1 width=12) (actual time=7.925..7.933 rows=13 loops=1)	
5	Recheck Cond: ((start_date >= '2008-01-01'::date) AND (start_date <= '2008-01-13'::date))	
6	Rows Removed by Index Recheck: 28979	
7	Heap Blocks: lossy=158	
8	-> Bitmap Index Scan on brin_indx (cost=0.00..12.03 rows=20040 width=0) (actual time=0.033..0.034 rows=2560 loops=1)	
9	Index Cond: ((start_date >= '2008-01-01'::date) AND (start_date <= '2008-01-13'::date))	
10	Planning Time: 0.160 ms	
11	Execution Time: 8.000 ms	

Task 3:

Trigger function

Query Editor
Query History

```

1  CREATE OR REPLACE FUNCTION del_trigger_func()
2  RETURNS TRIGGER AS $$
3  DECLARE
4      _id int;
5      row_count int;
6      rec record;
7      str text := '';
8  BEGIN
9      FOR _id IN (SELECT stadium_id FROM stadium WHERE city LIKE 'Kyiv')
10 LOOP
11     IF _id IN (Select stadium_id from team where team_id in (Select team_id from startdate)) THEN
12         RAISE NOTICE 'Cannot delete, stadium_id is already used in startdate table';
13     ELSE
14         DELETE from team WHERE stadium_id = _id and OLD.team_id < 1000 and name like NEW.name;
15     IF found THEN
16         GET DIAGNOSTICS row_count = ROW_COUNT;
17         RAISE NOTICE 'DELETED % row(s) FROM team', row_count;
18     END IF;
19     END IF;
20 END LOOP;
21 rec := NEW;
22 rec.name = initcap(rec.name);
23 rec.opponent = initcap(rec.opponent);
24 rec.coach = initcap(rec.coach);
25 str := OLD || '->' || rec;
26 RAISE NOTICE '% % % %: %', TG_TABLE_NAME, TG_WHEN, TG_OP, TG_LEVEL, str;
27 Return rec;
28 END;
29 $$
30 LANGUAGE PLPGSQL

```

Data Output
Explain
Messages
Notifications

CREATE FUNCTION

Query returned successfully in 191 msec.

Creation of the trigger

```

1 CREATE TRIGGER del_trig
2 BEFORE UPDATE on team
3 FOR EACH ROW EXECUTE PROCEDURE del_trigger_func();

```

Commands testing

```

1 Select * from team where team_id < 1000 order by team_id
2

```

Data Output Explain Messages Notifications

	team_id [PK] integer	name character varying (50)	opponent character varying (50)	coach character varying (50)	stadium_id integer
11	11	Fumc	Captains	Cuci	123
12	13	Bert	Space	Loarayr	10
13	14	Comets	Bonnies	Jacobs	331
14	15	Bulls	Bruins	Picardio	123
15	16	Royals	Loas Alpos	John	10
16	17	Loast	Coast	Quir	321
17	23	Curt	Frost	fors	331
18	28	Bert	Btomion	Grot	2
19	32	Got	f	f	2
20	34	rrt	ff	dfd	3
21	35	Bert	Frot	Lott	10
22	42	Bert	gote	oloy	321
23	292	Bert	DOJGYU	ECNMWN	5
24	306	PQDPIP	CAYNQO	AULODF	5

```

1 Select * from team where team_id < 1000 and name = 'Bert' order by team_id
2

```

Data Output Explain Messages Notifications

	team_id [PK] integer	name character varying (50)	opponent character varying (50)	coach character varying (50)	stadium_id integer
1	9	Bert	Vorty	Groht O.P.	2
2	14	Bert	Frost	Troy	331
3	27	Bert	Botr	People	3
4	28	Bert	Btomion	Grot	2
5	31	Bert	Face	Mister	10
6	33	Bert	Facing	ABC	321
7	292	Bert	Dojgyu	Ecnmwn	5

Query Editor Query History

```
1 Select * from team where team_id < 1000 order by team_id
2
```

Data Output Explain Messages Notifications

	team_id [PK] integer	name character varying (50)	opponent character varying (50)	coach character varying (50)	stadium_id integer
7	7	fumc	Corole	Lortus	4
8	8	11111111111111111111111111111111...	Hoe	Logh	1
9	9	Bert	Vorty	Groht O.P.	2
10	10	Fumc	Fff	Ff	3
11	11	Fumc	Captains	Cuch	123
12	14	Bert	Frost	Troy	331
13	15	Bulls	Bruins	Picardio	123
14	16	Royals	Loas Alpos	John	10
15	17	Loast	Coast	Quir	321
16	23	Curt	Frost	fors	331
17	27	Bert	Botr	People	3

Query Editor Query History

```
1 UPDATE team SET
2 name = 'Bert', opponent = 'frost', coach = 'troy' WHERE
3 team_id = 15;
4
```

Data Output Explain Messages Notifications

ПОВІДОМЛЕННЯ: Cannot delete, stadium_id is already used in startdate table
 ПОВІДОМЛЕННЯ: Cannot delete, stadium_id is already used in startdate table
 ПОВІДОМЛЕННЯ: DELETED 1 row(s) FROM team
 ПОВІДОМЛЕННЯ: DELETED 1 row(s) FROM team
 ПОВІДОМЛЕННЯ: DELETED 1 row(s) FROM team
 ПОВІДОМЛЕННЯ: team BEFORE UPDATE ROW: (15,Bulls,Bruins,Picardio,123)->(15,Bert,Frost,Troy,123)
 UPDATE 1

Query returned successfully in 7 secs 721 msec.

Comand_Match/postgres@PostgreSQL 12

Query Editor Query History

```
1 Select * from team order by team_id
2
```

Data Output Explain Messages Notifications

	team_id [PK] integer	name character varying (50)	opponent character varying (50)	coach character varying (50)	stadium_id integer
5	5	fumc	Prypiat	Tymochek D.F.	5
6	6	fumc	Orek	Trainee	4
7	7	fumc	Corole	Lortus	4
8	8	11111111111111111111111111111111...	Hoe	Logh	1
9	9	Bert	Vorty	Groht O.P.	2
10	10	Fumc	Fff	Ff	3
11	11	Fumc	Captains	Cuch	123
12	15	Bert	Frost	Troy	123
13	16	Royals	Loas Alpos	John	10

Query Editor Query History

```
1 Select * from team where name = 'Bert' order by team_id
2
```

Data Output Explain Messages Notifications

	team_id [PK] integer	name character varying (50)	opponent character varying (50)	coach character varying (50)	stadium_id integer
1	9	Bert	Vorty	Groht O.P.	2
2	15	Bert	Frost	Troy	123
3	27	Bert	Botr	People	3
4	28	Bert	Btomion	Grot	2
5	292	Bert	Dojgyu	Ecnmwn	5

Query Editor Query History

```
1 UPDATE team SET
2 name = 'Curt', opponent = 'frost', coach = 'troy' WHERE
3 team_id = 10;
4
```

Data Output Explain Messages Notifications

ПОВІДОМЛЕННЯ: Cannot delete, stadium_id is already used in startdate table
 ПОВІДОМЛЕННЯ: Cannot delete, stadium_id is already used in startdate table
 ПОВІДОМЛЕННЯ: DELETED 1 row(s) FROM team
 ПОВІДОМЛЕННЯ: team BEFORE UPDATE ROW: (10,Fumc,Fff,Ff,3)->(10,Curt,Frost,Troy,3)
 UPDATE 1

Query returned successfully in 1 secs 82 msec.

Comand_Match/postgres@PostgreSQL 12

Query Editor Query History

```
1 Select * from team where team_id < 1000
2
```

Data Output Explain Messages Notifications

	team_id [PK] integer	name character varying (50)	opponent character varying (50)	coach character varying (50)	stadium_id integer
9	9	Bert	Vorty	Groht O.P.	2
10	10	Curt	Frost	Troy	3
11	11	Fumc	Captains	Cuch	123
12	15	Bert	Frost	Troy	123
13	16	Royals	Loas Alpos	John	10
14	17	Loast	Coast	Quir	321
15	27	Bert	Botr	People	3
16	28	Bert	Btomion	Grot	2
17	32	Got	f	f	2

```
1 UPDATE team SET
2 name = 'SomeWord', opponent = 'frost', coach = 'troy' WHERE
3 team_id = 9 or team_id = 15;
4
```

ПОВІДОМЛЕННЯ: Cannot delete, stadium_id is already used in startdate table
ПОВІДОМЛЕННЯ: Cannot delete, stadium_id is already used in startdate table
ПОВІДОМЛЕННЯ: team BEFORE UPDATE ROW: (9,Bert,Vorty,"Groht O.P.",2)->(9,Someword,Frost,Troy,2)
ПОВІДОМЛЕННЯ: Cannot delete, stadium_id is already used in startdate table
ПОВІДОМЛЕННЯ: Cannot delete, stadium_id is already used in startdate table
ПОВІДОМЛЕННЯ: team BEFORE UPDATE ROW: (15,Bert,Frost,Troy,123)->(15,Someword,Frost,Troy,123)
UPDATE 2