1 - билет

source: [Postgre](https://blog.skillfactory.ru/glossary/postgresql/)

**What is PostgreSQL, and how does it differ from other relational database systems?**

PostgreSQL is a free database management software that allows you to store and organize large amounts of information. It is known for its reliability and its ability to strictly follow the rules of SQL, the language for working with databases.

The decision between Postgres and other databases will be determined by your organization's unique needs and requirements. Postgres is a strong choice for applications that require advanced query functionality, support for complex data structures, or the ability to handle large amounts of data.

**​​Oracle vs. PostgreSQL:**

Oracle is a powerful database management system used by large enterprises and is known for its reliability. PostgreSQL is also reliable but differs from Oracle in its openness and lower cost of ownership. PostgreSQL offers many features and is a more affordable choice for many projects due to its flexibility and community support.

**MySQL vs. PostgreSQL:**

MySQL is another popular open-source database management system, but it has a proprietary license as opposed to the freer license of PostgreSQL. PostgreSQL is considered more powerful and feature-rich, especially when dealing with complex queries and large amounts of data.

**MongoDB vs PostgreSQL:**

MongoDB is a NoSQL database management system designed to handle large amounts of data in a flexible JSON-like format. MongoDB is easier to set up and use, but is not as functional and powerful as PostgreSQL, especially when it comes to complex queries and transactions.

2 - билет

**Can you explain the key features of PostgreSQL?**

**Extensibility:**

You can add new functions and data types and write code in programming languages such as Python or JavaScript. This allows you, for example, to create special functions to process data directly in the database.

**SQL standards compliance:**

PostgreSQL supports many advanced SQL functions, including window functions for calculating statistics within data groups and recursive queries, which is important for complex analytic queries.

**Support for a variety of data types:**

In addition to the usual numbers and text, PostgreSQL can handle JSON, XML, and arrays, making it ideal for applications that handle a variety of data.

**Powerful data management tools:**

PostgreSQL offers advanced indexing techniques and full-text search, which speeds up data retrieval and makes it easier to build systems with complex text queries.

**High performance and reliability:**

Data replication and recovery systems provide high availability and reliability. In the event of failures, you can quickly switch to a backup server or restore data to a point in time.

These features make PostgreSQL an attractive choice for applications ranging from websites to large analytics platforms.

<https://www.quest.com/learn/what-is-postgresql.aspx>

3 - билет

**What are the different data types supported by PostgreSQL?**

PostgreSQL supports many different data types, making it a very flexible tool for working with a variety of data. Here are some of the main data types available in PostgreSQL:

**1. \*\*Numeric data types:\*\***

- `SMALLINT`, `INTEGER`, `BIGINT` - integers of different lengths.

- `DECIMAL`, `NUMERIC` - numbers with fixed precision and scale.

- `REAL`, `DOUBLE PRECISION` - floating point numbers.

- `SERIAL`, `BIGSERIAL` - autoincremental numbers for automatic generation of unique values.

**2. \*\*Text data types:\*\***

- `CHAR(n)`, `VARCHAR(n)` - strings of fixed and variable length.

- `TEXT` - text with no length limit.

**3. \*\*Date and time:\*\***

- `DATE` - date only.

- `TIME`, `TIME WITH TIME ZONE` - time with or without time zone.

- `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE` - date and time with or without time zone.

- `INTERVAL` - time interval.

**4. \*\*Logic type:\*\***

- `BOOLEAN` - logical data type (true/false).

**5. \*\*Data types for working with geometry:\*\***

- `POINT`, `LINE`, `LSEG`, `BOX`, `PATH`, `POLYGON`, `CIRCLE` - various geometric shapes.

**6. \*\*Network addresses:\*\***

- `INET` - IP addresses.

- `CIDR` - network addresses.

- `MACADDR`, `MACADDR8` - MAC addresses.

**7. \*\*JSON, XML and others:\*\***

- `JSON`, `JSONB` - for storing JSON data. `JSONB` provides more efficient storage in binary format.

- `XML` - for storing XML data.

- `HSTORE` - for storing key-value sets.

4 - билет

**Как PostgreSQL управляет параллелизмом и транзакциями?**

<https://sql-ex.ru/blogs/?/Parallelizm_bazy_dannyh_v_PostgreSQL.html>

Parallelism and transaction are two important concepts in database management, especially when many users are working on the same database at the same time.

Parallelism

Parallelism in the context of databases refers to the ability of a system to process multiple transactions simultaneously. This helps improve performance and speed of data processing by allowing users to perform different tasks (both reading and writing data) without waiting for each other to complete.

Transaction

A transaction is a sequence of database operations that is treated as a whole. A transaction must be completely executed or not executed at all. This ensures data integrity by ensuring that the database does not end up in an inconsistent state after failures or errors.

Managing concurrency and transactions in PostgreSQL

PostgreSQL manages concurrency and transactions through several mechanisms:

Multi-version concurrency control (MVCC):

MVCC allows multiple users to read and modify data simultaneously without locking. Instead of locking data for the duration of a read operation, PostgreSQL allows data to be read as it was when the operation started, even if other users make changes. This reduces waits and improves system performance.

Write-Ahead Logging (WAL):

MENTION ABOUT ACID!!!!

5 - билет

**Can you describe the architecture of PostgreSQL?**

The architecture of PostgreSQL is an important aspect of its powerful and reliable performance as a database management system. It's built as a client-server system and offers robust features that support transactions, concurrency, reliability, and extensibility. Let’s delve into the main components of PostgreSQL’s architecture:

### **1. Server and Client Model**

PostgreSQL operates on a client-server model, where the server handles all the primary tasks related to data management, transaction processing, and storage, while clients connect to the server to send commands (queries) and receive responses.

* Server: The PostgreSQL server, also known as postgres, is a collection of processes handling database operations. The main daemon process listens for incoming connections from client applications and spawns a new backend process for each client connection.
* Client: Clients can be any application that communicates with the PostgreSQL server using the PostgreSQL Frontend/Backend Protocol. These include command-line tools like psql, libraries from various programming languages, or specialized graphical tools like pgAdmin.

### **2. Processes**

* Postmaster: The primary process that initializes the system at startup and manages client connections, launching new backend processes as needed.
* Backend Processes: For each client connection, a separate backend process is created. This process handles all the tasks of query parsing, analysis, execution, and access to data. Each backend process is isolated; thus, they do not share memory directly.

### **3. Memory Architecture**

* Shared Memory: Used to store data that is shared among all server processes, including the buffer cache, lock tables, and write-ahead logging (WAL) buffers.
* Local Memory: Each server process has local memory for sorting and other operations.

### **4. Storage System**

* Data Files: All data is stored in files in a file system. Each table and index is stored in separate files.
* Write-Ahead Logging (WAL): This feature ensures data integrity by recording changes to a log file before any changes (or "commits") are written to the database. This allows recovery in the event of a crash.
* Transaction Log: The WAL files are part of the broader transaction log, which plays a critical role in recovery and replication.

### **5. Transaction Management**

* MVCC (Multi-Version Concurrency Control): PostgreSQL uses MVCC to manage concurrent access to the database. Instead of locking data during a read, which blocks writes, PostgreSQL allows multiple versions of data to coexist, enabling reads to fetch the version of data that existed when their transaction started.

### **6. Query Processor**

* Parser: Breaks down SQL queries into parse trees.
* Rewriter: Applies rewrite rules to the parse trees.
* Planner/Optimizer: Takes a parse tree and transforms it into a query plan, which is the most efficient way to execute the query.
* Executor: Executes the query plan using the underlying data.

### **7. Extensibility**

* Custom Functions: Users can define their functions and procedures in various programming languages.
* Data Types: It’s possible to define new data types.
* Operators: New operators for existing or new data types can be created.
* Foreign Data Wrappers: These allow PostgreSQL to query external data sources, such as other SQL databases or flat file systems.

### **8. Networking**

* Listening: PostgreSQL listens on a specified port for connections from client applications.
* Security: Connections can be secured using SSL, and various authentication methods can be configured, including password-based and certificate-based authentication.

This layered and modular architecture not only provides high performance and scalability but also makes PostgreSQL highly flexible and extensible, which is why it's favored for everything from small applications to large enterprise systems.

РАССКАЗАТb ПРО ENTITY /// ONE TO ONE /// ONE TO MANY

6 - билет

**What are some common use cases for PostgreSQL?**

Common PostgreSQL use cases

**Web applications**: often used as a database server for dynamic websites and web applications.

**Geospatial databases**: with PostGIS support, PostgreSQL becomes a powerful geospatial database.

**Analytical applications**: its sophisticated query and aggregation capabilities make it ideal for data analytics.

**Financial systems**: due to its stringent requirements for transactional integrity and security.

**Content management systems**: flexibility to manage different types of data and large volumes of information.

**MIRAS**

**7 - билет**

**How does PostgreSQL handle indexing, and what types of indexes does it support?**

PostgreSQL uses various types of indexes to improve query performance, including B-tree, Hash, GiST, SP-GiST, GIN, and BRIN indexes. B-tree indexes are most commonly used for general-purpose indexing, while the other types have specific use cases such as full-text search (GIN), range queries (BRIN), and spatial data (GiST, SP-GiST).

B-tree index is the most commonly used type of index to efficiently store and retrieve data in PostgreSQL. It's the default index type. Whenever we use the CREATE INDEX command without specifying the type of index we want, PostgreSQL will create a B-tree index for the table or column.

**8 - билет**

**What is the role of Vacuum in PostgreSQL, and how does it work?**

VACUUM reclaims storage occupied by dead tuples. In normal PostgreSQL operation, tuples that are deleted or obsoleted by an update are not physically removed from their table; they remain present until a VACUUM is done. Therefore it's necessary to do VACUUM periodically, especially on frequently-updated tables.

**9 - билет**

**How can you optimize PostgreSQL queries for performance?**

Some strategies for optimizing PostgreSQL queries include proper indexing, writing efficient SQL queries, avoiding unnecessary joins and subqueries, using EXPLAIN to analyze query plans, optimizing configuration parameters like memory settings and parallelism, and considering denormalization for read-heavy workloads.

**Detailed info:** [**How to Optimize PostgreSQL Queries | by NodeTeam**](https://nodeteam.medium.com/how-to-optimize-postgresql-queries-226e6ff15f72)

**10 - билет**

**What are some best practices for database design in PostgreSQL?**

Best practices for database design in PostgreSQL include normalizing your database schema to reduce redundancy, using appropriate data types, defining constraints to enforce data integrity, partitioning large tables when necessary, indexing columns that are frequently queried, and considering the performance implications of your design choices.

**Detailed info:** [**Best Practices for PostgreSQL Database**](https://www.e2enetworks.com/blog/best-practices-for-postgresql-database)

**11 - билет**

**Can you explain the role of roles and privileges in PostgreSQL?**

**ТУТ МОЖНО ЕБАШИТЬ ПРО ТО КАКОЙ ПОСТГРЕ АХУЕННЫЙ (НЕ ОБЯЗАТЕЛЬНО ПРО ВНУТРЕННИЕ РОЛИ И ПРИВЕЛЕГИИ)**

Roles, objects, and privileges. Like other access control mechanisms, PostgreSQL's access control can be explained like “Role X is allowed to do Y on object Z”. Here, roles are users and groups, objects are databases, tables, etc., and Privileges include SELECT, INSERT, UPDATE, DELETE, and more.

**More info:** [**PostgreSQL roles and privileges explained - Aviator Blog**](https://www.aviator.co/blog/postgresql-roles-and-privileges-explained/#:~:text=Roles%2C%20objects%2C%20and%20privileges,actions%20like%20SELECT%20or%20INSERT%20)**.**

**12 - билет**

**What is the purpose of schemas in PostgreSQL, and how are they used?**

Schemas in PostgreSQL are logical containers that allow you to organize database objects such as tables, stored procedures, functions, etc. Schemas provide a way to rationally group related objects and separate them based on their functionalities or business domains. In other words, it is a conceptual space that contains named database objects such as tables, stored procedures, etc.

More info: [Schemas in PostgreSQL](https://www.dbvis.com/thetable/schemas-in-postgresql/#:~:text=Schemas%20in%20PostgreSQL%20are%20logical,their%20functionalities%20or%20business%20domains).

**13 билет How does PostgreSQL handle JSON and JSONB data types? Как PostgreSQL обрабатывает типы данных JSON и JSONB?** [**источник**](https://postgrespro.ru/docs/postgresql/15/datatype-json)

In PostgreSQL, the JSON and JSONB data types allow storing and operating on data in the JSON (JavaScript Object Notation) format. Both types are textual values, but they have some differences in how data is stored and processed.

**1. JSON:**

- JSON (JavaScript Object Notation) is stored as text.

- JSON storage preserves the exact structure of the JSON document, including spaces and field order.

- JSON can contain duplicate keys, but they may be lost when converting to JSON.

- JSON supports fast read and write operations but may be less efficient for queries such as key lookup or filtering.

**2. JSONB**:

- JSONB (JSON Binary) is stored in a binary format.

- Unlike JSON, JSONB does not preserve the exact structure of the JSON document, which provides more compact storage and more efficient comparison and search operations.

- JSONB does not support duplicate keys.

- Storage in JSONB format requires more computational resources for conversion to binary format during data insertion and update, but this is offset by more efficient query operations.

**3. Indexing and Operations:**

- PostgreSQL supports indexing for JSONB data fields, providing efficient execution of queries involving key or value lookup in JSONB data.

- There are many operators and functions for working with JSON and JSONB data, including access to JSON object and array elements, merge operations, aggregation, etc.

**4. Conversion Between JSON and JSONB:**

- PostgreSQL provides functions for converting data between JSON and JSONB types, providing flexibility when working with data in different formats. 5. Constraints:

- JSON and JSONB can contain data of various types and structures, but they cannot contain certain data types such as functions and PostgreSQL-specific data types like bytea.

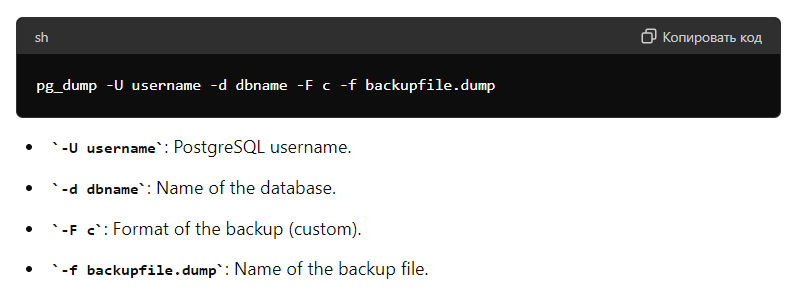
- Constraints on JSON and JSONB data types are determined by the JSON and JSONB standards, which should be considered when designing the data schema.

Overall, JSONB is usually the preferred choice for storing and operating on JSON data in PostgreSQL due to its more compact storage format and the efficiency of search and comparison operations. However, if it is necessary to preserve the exact structure of the JSON document, the JSON data type can be used.

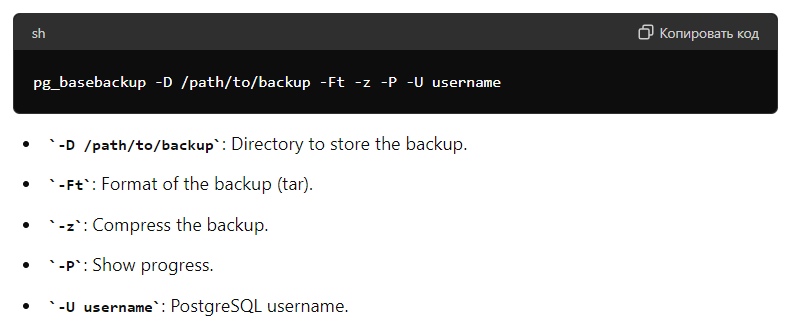
**14 билет. Can you explain the process of backing up and restoring a PostgreSQL database? Можете ли вы объяснить процесс резервного копирования и восстановления базы данных PostgreSQL?** [**источник**](https://postgrespro.ru/docs/postgresql/14/backup) **/** [**другой источник**](https://habr.com/ru/companies/otus/articles/710094/)

**Резервное копирование** [Резервное копирование является одной из основных задач администрирования баз данных. Отсутствие бэкапа может привести к катастрофическим последствиям при эксплуатации базы данных. Однако недостаточно только настроить регулярное создание резервных копий, необходимо также регулярно проверять созданные копии на способность к восстановлению БД. В этой статье мы поговорим от том, как правильно настраивать резервное копирование в БД PostgreSQL.]

1. **pg\_dump**: A command-line utility for creating logical backups of PostgreSQL databases (structure and data).

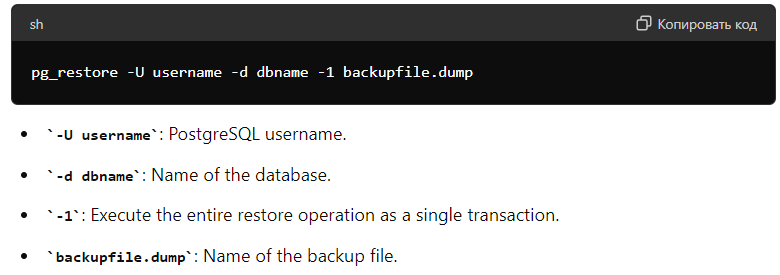
****

2. **pg\_basebackup**: Used for creating physical backups of the entire PostgreSQL cluster.



**Restore** [Резервное копирование без регулярной проверки сделанных бэкапов это не более чем хранение данных. Только регулярная проверка созданных бэкапов позволяет быть уверенными в том, что наши резервные копии данных не подведут в нужный момент.]

1. **pg\_restore**: Used to restore data from logical backups created by pg\_dump.



1. **Restoring from a physical backup**:
   * Stop the PostgreSQL server.
   * Remove old data from the PostgreSQL data directory.
   * Extract the backup into the data directory.
   * Start the PostgreSQL server.

**15 билет. What tools and utilities are available for monitoring and managing PostgreSQL databases? Какие инструменты и утилиты доступны для мониторинга и управления базами данных PostgreSQL?** [**тык**](https://habr.com/ru/companies/otus/articles/690266/)

#### **pgAdmin (must have)**

pgAdmin is a widely used graphical tool for administering and developing PostgreSQL databases. It offers an intuitive user interface for managing database objects, running SQL queries, and visualizing data. Key features include:

* **Database management**: Create, modify, and delete databases, tables, indexes, and other database objects.
* **Query tool**: Execute and debug SQL queries with a powerful query editor.
* **Data visualization**: View and edit table data in a spreadsheet-like format.
* **Backup and restore**: Perform backups and restores using an easy-to-use interface.

#### **psql**

psql is a command-line interface for interacting with PostgreSQL. It allows users to execute SQL commands, manage database objects, and perform administrative tasks. Key features include:

* **Interactive terminal**: Run SQL queries and see immediate results.
* **Scripting**: Execute SQL scripts and automate tasks using shell scripts.
* **Database administration**: Manage roles, permissions, and perform maintenance tasks.
* **Customization**: Configure environment settings and create custom commands.

#### **PostgreSQL Enterprise Manager**

PostgreSQL Enterprise Manager (PEM) by EnterpriseDB is a comprehensive tool for monitoring, administration, and management of PostgreSQL databases. Key features include:

* **Monitoring**: Real-time monitoring of database performance and resource usage.
* **Alerts and notifications**: Set up alerts for critical events and receive notifications.
* **Performance tuning**: Analyze and optimize query performance.
* **Administration**: Manage databases, users, roles, and permissions through a unified interface.

#### **Nagios**

Nagios is a powerful monitoring system that can be extended with plugins to monitor PostgreSQL databases. Key features include:

* **Customizable plugins**: Use existing plugins or create custom ones to monitor PostgreSQL metrics.
* **Alerting**: Set up alerts for performance issues, downtime, and other critical events.
* **Dashboards**: Create visual dashboards to monitor the health of your PostgreSQL servers.
* **Integration**: Integrate with other tools and services for comprehensive monitoring.

#### **Prometheus + Grafana**

Prometheus and Grafana are a popular combination for collecting and visualizing metrics. Prometheus collects and stores metrics, while Grafana provides a powerful dashboarding and visualization layer. Key features include:

* **Metric collection**: Gather detailed metrics about PostgreSQL performance, resource usage, and more.
* **Alerting**: Set up alert rules based on Prometheus metrics and receive notifications.
* **Dashboards**: Create customizable dashboards in Grafana to visualize key metrics.
* **Scalability**: Handle large-scale monitoring across multiple PostgreSQL instances.

#### **pg\_stat\_statements**

pg\_stat\_statements is a PostgreSQL extension that collects statistics on SQL queries executed in the database. Key features include:

* **Query tracking**: Track execution statistics for all SQL queries, including frequency, duration, and resource usage.
* **Performance analysis**: Identify slow-running queries and optimize them.
* **Historical data**: Keep historical data on query performance for trend analysis.
* **Integration**: Integrate with other monitoring tools to enhance performance monitoring.

#### **pgBadger**

pgBadger is a tool for analyzing PostgreSQL log files and generating detailed reports. It helps database administrators understand the behavior and performance of their PostgreSQL databases. Key features include:

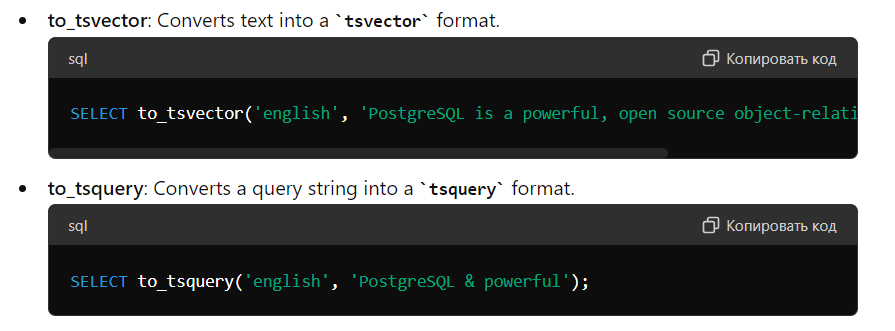
* **Log parsing**: Parse PostgreSQL log files to extract useful information.
* **Reports**: Generate comprehensive reports on query performance, errors, and resource usage.
* **Visualization**: Visualize log data in an easy-to-understand format.
* **Customization**: Configure report generation to focus on specific metrics and time periods.

**16 билет. How does PostgreSQL handle full-text search and what are its capabilities in this area? Как PostgreSQL работает с полнотекстовым поиском и каковы его возможности в этой области?** [**тык**](https://habr.com/ru/companies/otus/articles/770674/) **и** [**тут тоже**](https://yulia633.github.io/fts-laravel/)

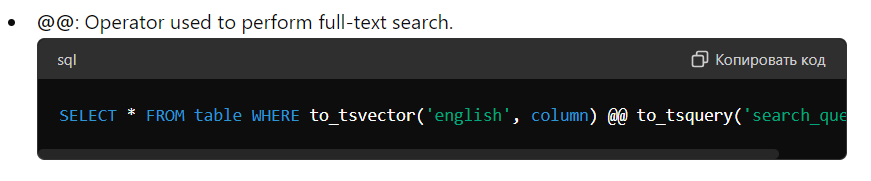
PostgreSQL provides powerful tools for implementing full-text search, enabling efficient search capabilities within text data stored in the database. Let's explore how this works and the capabilities PostgreSQL offers.

#### **Basic Concepts**

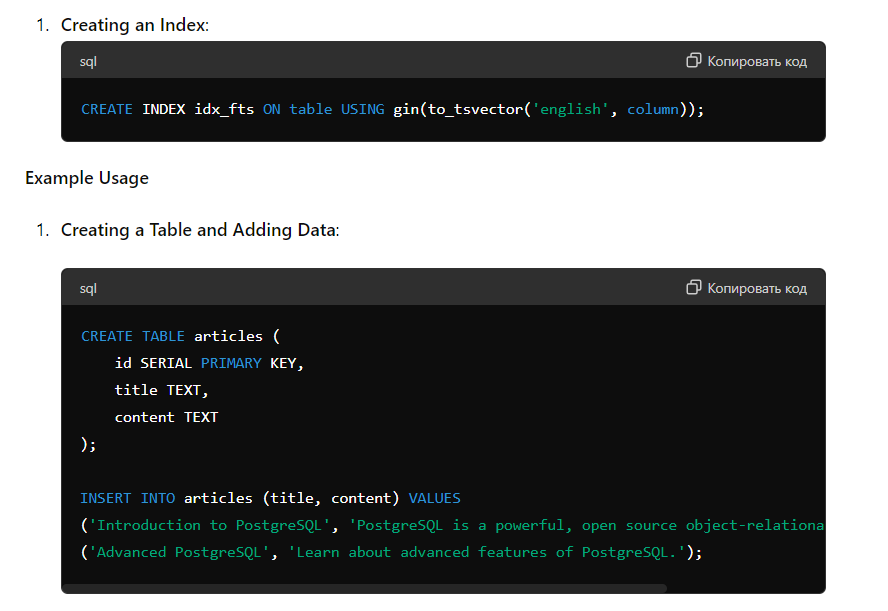
1. **Data Types**:
   * **tsvector**: A data type used to store preprocessed text for full-text search. It stores a set of lexemes (key words) with their positions in the text.
   * **tsquery**: A data type used to represent search queries.
2. **Conversion Functions**:

****

1. **Search Operators**:

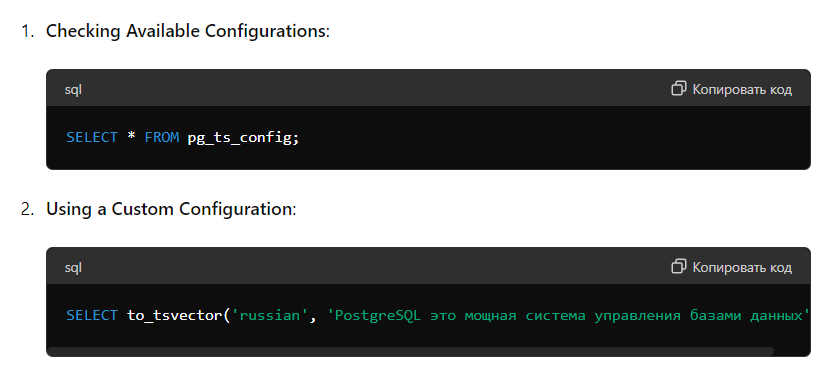
****

#### **Indexing**

To speed up full-text searches, PostgreSQL supports GIN (Generalized Inverted Index) and GiST (Generalized Search Tree) indexes.****

#### **Full-Text Search Configurations**

PostgreSQL supports various configurations for full-text search that include settings for different languages. For instance, you can use an English configuration that accounts for morphology and stemming (reducing words to their root form).

****

### **PostgreSQL Full-Text Search Capabilities**

### 

Full-text search in PostgreSQL is a powerful and flexible tool that allows for fast and accurate searches within text data. With support for various languages, result ranking, and indexing capabilities, PostgreSQL provides all the necessary means to implement efficient full-text search functionality.

**17 билет. What are some security features and mechanisms provided by PostgreSQL? Какие функции и механизмы безопасности предоставляет PostgreSQL?** [**почитать**](https://habr.com/ru/articles/550882/)

**Security at the network level**

1. **Firewalls**

Ideally, your PostgreSQL server should be completely isolated and not allow any incoming connections, SSH or psql. Unfortunately, PostgreSQL does not support a standard configuration of this type. The next way that you can take to increase the security of the database server is to block access to the node on which the database is running at the port level using a firewall

1. **Listening to addresses**

It is a good practice to limit the addresses that the server listens to for client connections using the listen\_addresses parameter of the configuration file. If the node running PostgreSQL has multiple network interfaces, use this option to make sure that the server listens only to those interfaces through which clients will connect to it:

**Safety at the transport level**

Since most of the world wide web is switching to HTTPS, there are few excuses not to use strong encryption for database connections. PostgreSQL supports TLS (which is still called SSL in documentation, configuration and CLI for legacy reasons) and allows it to be used for authentication of both the server and the client.

1. **Server-side TLS**

To authenticate the server, you must first obtain a certificate that the server will provide to connecting clients. Let's Encrypt makes it easier to obtain free X.509 certificates, for example, using the CLI certbot tool

1. **Client TLS**

Client authentication by certificate allows the server to verify the identity of the connecting user, confirming that the X.509 certificate submitted by the client is signed by a trusted certificate authority (CA). It is recommended to use different certificate authorities to issue certificates to the client and server, so let's create a client CA and use it to sign the client certificate

1. **TLS Configuration**

Having put all the pieces together, you can now configure the PostgreSQL server to accept TLS connections.

**Database-level security**

1. **Roles**

So far, we've been looking at how to protect the PostgreSQL database server from unauthorized network connections, use encryption, and make sure that the server and clients can trust each other through mutual TLS authentication. Another piece of the puzzle is to figure out what users can do and what they have access to after connecting to the database and verifying their identity. This procedure is usually called authorization.

1. **Row-level security**

One of the most advanced features of the PostgreSQL privilege system is row-level security, which allows you to grant privileges to a subset of rows in a table. This includes both rows that can be queried using the **SELECT** statement, and rows that can be the results of **INSERT**, **UPDATE**, and **DELETE** statements. To start using row-level security, you need two things: enable it for the table and define a policy that will control row-level access.

1. **Audit**

So far, we have mainly talked about preventive security measures. Following the cornerstone principles of security — defense in depth, we looked at how they overlap to slow down the progress of a hypothetical attacker through the system.

Maintaining a detailed audit trail is one of the system security options that is often overlooked. Monitoring network-level or node-level access for your database server is beyond the scope of this article, but let's see what capabilities we have when it comes to the PostgreSQL server itself.

**Conclusion**

As with any system designed with security in mind, proper protection of access to a database instance requires protective measures at several levels of the stack.

**18 билет. Can you discuss the differences between PostgreSQL's procedural languages like PL/pgSQL, PL/Python, and PL/Java? Можете ли вы рассказать о различиях между процедурными языками PostgreSQL, такими как PL/pgSQL, PL/Python и PL/Java?**

PostgreSQL supports several procedural languages that allow you to write functions and triggers for execution inside a database. Each of these languages has its own characteristics and purpose.

### **PL/pgSQL**

* **Description**: A procedural language embedded in PostgreSQL, designed for writing functions and triggers using SQL with procedural extensions.
* **Features**:
  + **SQL Integration**: Easily combines SQL queries with procedural constructs like loops and conditions.
  + **Performance**: Typically faster since it is a built-in language.
  + **Ease of Use**: Ideal for users familiar with SQL who want to extend its capabilities.
  + **Use Cases**: Commonly used for business logic, data integrity checks, and other operations directly within the database.

### **PL/Python**

* **Description**: A procedural language that allows the use of Python for writing functions and triggers in PostgreSQL.
* **Features**:
  + **Python Power**: Leverages Python's rich set of libraries, enhancing data processing and integration capabilities.
  + **Flexibility**: Suitable for complex calculations and handling large data volumes.
  + **High-Level Syntax**: Provides high-level syntax and data structures of Python.
  + **Use Cases**: Useful for tasks requiring advanced mathematical computations, data analysis, and integration with external systems where PL/pgSQL might be limited.

### **PL/Java**

* **Description**: A procedural language that enables the use of Java for writing functions and triggers in PostgreSQL.
* **Features**:
  + **Java Power**: Utilizes the capabilities of the Java platform, including extensive libraries and frameworks.
  + **Cross-Platform**: Known for running on various platforms without code changes.
  + **Integration**: Easily integrates with other Java applications and systems.
  + **Use Cases**: Beneficial for enterprise applications requiring Java capabilities such as multithreading, networking, and other specific tasks.

### **Comparison**

* **Ease of Learning**:
  + **PL/pgSQL**: Easier for those familiar with SQL.
  + **PL/Python**: Easy for Python users and those utilizing its libraries.
  + **PL/Java**: Requires knowledge of Java and its ecosystem.
* **Performance**:
  + **PL/pgSQL**: Typically the fastest for typical database tasks since it is built into PostgreSQL.
  + **PL/Python and PL/Java**: May be slower due to overhead from interacting with external languages, but can be optimized for specific tasks.
* **Capabilities**:
  + **PL/pgSQL**: Best for tasks closely related to SQL.
  + **PL/Python and PL/Java**: Provide extended capabilities for data processing, integration, and specialized computations.

The choice between these languages depends on specific tasks, programming preferences, and required functionality

**Айзира**

19-Билет.

**How does PostgreSQL handle data replication and high availability?**

[Источник Доп. Чтение](https://vuyisile.com/high-availability-in-postgresql-replication-with-docker/)

High Availability in the context of database systems refers to the capability of a system to minimize or eliminate downtime in order to remain accessible and operational without interruptions, ensuring continuous service delivery to users and applications.

To build highly available database systems, database administrators have to implement one or more of the solutions listed below:

**Standby Replication**

Standby replication means making a copy of the main PostgreSQL database on another server. This copy, called a replica or standby server, stays updated with all the changes happening on the main server. It's like having a backup that's always in sync with the main one. If something happens to the main server and it goes down, the standby server can step up and take over as the main server, keeping things running smoothly. There are different ways to do this replication, like log shipping, streaming replication, cascading replication, and logical replication. PostgreSQL keeps track of all the changes made to the database in something called a Write Ahead Log (WAL). Every time a transaction is saved to the database on the main server, it creates a WAL record. The standby servers use these WAL records to stay in sync with the main server, so they always know what's going on.

20-Билет.

**What are the advantages and disadvantages of using PostgreSQL as a database system?**  [ИСТОЧНИК ДОПОЛНИТЕЛЬНО ПРОЧИТАТЬ](https://www.hackertrail.com/talent/database/postgresql-interview-questions-and-answers/#advantages-anddisadvantages-postgresql-database)

Advantages:

* Open-source provides free access to users for implementing and modifying it as per their need.
* High-risk tolerant database due to WAl and PITR implementations.
* Easy to learn and use.
* Supports dynamic web applications.
* Supports processing of complex data types like graphical data, geographical data, etc.
* User-defined data types, functions, triggers can be created.

Disadvantages:

* Performance is slower when compared to MySQL.
* The creation of database replication is a little complex.
* It is not maintained by a single company hence it is less popular.

21-Билет.

**Can you explain the role of triggers in PostgreSQL and provide examples of their use?** [Источник ДОПОЛНИТЕЛЬНО ПРОЧИТАТЬ](https://medium.com/simform-engineering/what-when-and-how-of-database-triggers-using-postgresql-f4464bd969d4)

TRIGGER as the word is self-explanatory, triggers actually triggers something. I know it sounds dumb but it is what it is.

DB triggers are the database objects which are associated with tables. A trigger is a stored procedure which gets executed before/after the occurrence of some event. In terms of DB, events can be insertion to the table, deletion from the table, modification of the records of the table.

Triggers are generally used for the tasks that need to be executed no matter what, it needs to be triggered and the database administrator/database designer does not have to worry about it. But it can be used for some NOT SO usual use cases as well.

An example. Let’s say you developed around 30 separate APIs for some WWW project. After halfway through the project you came across the module which described itself as “User Activity”. You need to track when the users updated their profile pictures, updated the password of their accounts, when did they get notifications, what was the content of the notifications. In summary, you need to track when most of those APIs got served. What your first approach might be to solve this problem is by tracking each activity followed by updating each of those APIs. What tedious and redundant work is that! What can be an alternative solution is to create a trigger for those database tables which are being interacted by those APIs underneath. Because in the end what APIs are really doing is manipulating/adding/deleting the data from the database.

22-Билет.

**How does PostgreSQL handle locking mechanisms, and what are the implications for concurrent access?** [Источник](https://medium.com/@sonishubham65/understanding-postgres-locks-and-managing-concurrent-transactions-1ededce53d59) [Дополнительно прочитать](https://habr.com/ru/companies/postgrespro/articles/462877/)

PostgreSQL utilizes a multi-level locking mechanism to control access to data at various granularities, including tables, rows, and individual data pages. These locks prevent conflicting operations from occurring simultaneously, thus ensuring data integrity. Two fundamental lock types are RowShareLock and RowExclusiveLock.

* RowShareLock: Also known as a shared lock, it allows multiple transactions to read a row concurrently but prevents any transaction from modifying it. This lock type ensures that read operations do not interfere with each other.
* RowExclusiveLock: This exclusive lock mode allows a transaction to both read and modify a row exclusively. It prevents other transactions from accessing the same row for reading or writing until the lock is released.

When accessing data concurrently in PostgreSQL, you should consider how locking mechanisms affect the ability of other transactions to perform operations while one transaction has already locked the data. Here's a more detailed explanation:

Hard locks:

* When a transaction sets a hard lock on data (such as a row in a table), it prevents any other operations (reads or writes) on that data until the lock is released. This means that other transactions attempting to access data locked by a hard lock will wait until the lock is released.

Soft locks:

* Unlike hard locks, soft locks allow simultaneous read access to data by other transactions, but not for modification. For example, if transaction A has set a soft lock on a row, transaction B can read that row but cannot modify it until transaction A releases the lock.

23-Билет.

**What is the role of extensions in PostgreSQL, and can you provide examples of popular extensions?**  [Источник ДОП. ПРОЧИТАТЬ](https://airbyte.com/data-engineering-resources/postgresql-extensions)

PostgreSQL extensions are add-on modules that enhance the functionality of the database solution. They provide additional features, data types, functions, and operators that are not present in the core Postgres system. The extension architecture in PostgreSQL allows developers to create and package their features as self-contained units, making it easy to install, update, and manage extensions independently of the main PostgreSQL installation. This modular approach ensures that the core database remains lean and efficient while allowing users to extend its capabilities to match their needs.

1. **PostGIS** is a powerful open-source extension that enables PostgreSQL to handle geographic objects and spatial data. This eliminates the need for separate specialized systems and allows location queries to be run in SQL. PostGIS extends PostgreSQL by introducing new data types and functions for storing, querying, and analyzing spatial data, such as points, lines, polygons, and more complex geometries.
2. **pg\_stat\_statements** is a built-in PostgreSQL extension that provides a way to collect and track statistics about SQL statements executed in the database. It records information like the total execution time, number of calls, and number of rows returned for each SQL statement. This extension is a valuable tool for developers to analyze and optimize queries. It makes optimizing the database system and improving the overall application performance easier.

24-Билет.

**How does PostgreSQL support geographical data and spatial queries?** [Источник](https://mappitall.com/blog/postgid-geometry-vs-geography#:~:text=You%20need%20PostGIS%20to%20store%2C,most%20out%20of%20your%20data)  [Доп Чтение](https://www.abstractapi.com/guides/postgres-geolocation)

Postgresql databases support geolocation queries through the use of stored geographical objects and the handy **PostGIS extension**, a spatial database extender.

PostGIS is needed to store, manage, and analyze spatial data in a PostgreSQL database. PostGIS provides functions and data types that allow you to store and manipulate geospatial data and perform spatial queries and analyses. Some common use cases for PostGIS include mapping, geocoding, routing, and spatial analysis. When working with data in a PostgreSQL database, PostGIS is an essential tool to help get the most out of your data.

**25. Can you discuss the role of foreign keys and referential integrity in PostgreSQL?  
Можете ли вы рассказать о роли внешних ключей и ссылочной целостности в PostgreSQL?** [**Source:**](https://www.postgresql.org/docs/current/ddl-constraints.html#:~:text=A%20foreign%20key%20constraint%20specifies,integrity%20between%20two%20related%20tables.)

Foreign keys in PostgreSQL enforce relationships between tables by linking a column in one table to the primary key in another. They maintain referential integrity, ensuring that data remains consistent. PostgreSQL offers options like CASCADE and SET NULL to manage actions like updates or deletions on referenced rows. This prevents orphaned rows or invalid references, maintaining relational integrity and facilitating easier database management.

**26. How does PostgreSQL handle data import and export from/to different formats?  
Как PostgreSQL обрабатывает импорт и экспорт данных из/в различные форматы?** [**Доп. Инфа**](https://codedamn.com/news/sql/importing-and-exporting-data-in-postgresql)

PostgreSQL handles data import/export through SQL dumps, CSV files with COPY command, and other formats like plain text and tab-separated values. It also supports Foreign Data Wrappers for accessing external data sources. Additionally, GUI tools like pgAdmin and third-party tools offer further options for these tasks. It is generally safer to import data containing embedded line end characters using the text or binary formats rather than CSV.

**27. What are some common pitfalls or challenges when working with PostgreSQL, and how can they be mitigated?  
Каковы некоторые распространенные подводные камни или проблемы при работе с PostgreSQL и как их можно устранить?**

Working with PostgreSQL can present challenges such as performance issues, concurrency problems, data integrity concerns, backup and recovery difficulties, upgrade complexities, security risks, and scaling challenges. Mitigate these challenges by optimizing queries and indexes, managing transactions effectively, implementing proper backups and disaster recovery plans, carefully planning upgrades, maintaining strong security measures, and scaling appropriately based on workload requirements.

**Common pitfalls or challenges when working with PostgreSQL include:**

**Performance Issues:** Poorly optimized queries, lack of indexes, and inefficient database design can lead to performance issues.

***Problem Solving:*** *Regularly analyze query performance, create appropriate indexes, optimize database schema design, and use EXPLAIN to understand query execution plans.*

**Concurrency Problems:** High concurrency can lead to contention and locking issues, impacting performance and scalability.

***Problem Solving:*** *Implement proper transaction isolation levels, use row-level locking, minimize long-running transactions, and consider partitioning for large tables*.

**Data Integrity:** Ensuring data integrity requires careful management of constraints and transactions to prevent inconsistencies.

***Problem Solving:*** *Use constraints (e.g., foreign keys, unique constraints), transactions, and referential integrity mechanisms. Regularly validate and clean data.*

**Backups and Disaster Recovery:** Inadequate backup and recovery strategies can result in data loss and downtime during failures.

***Problem Solving:*** *Implement regular backups using tools like pg\_dump or continuous archiving (WAL). Test backups regularly and have a robust disaster recovery plan.*

**Upgrades and Maintenance:** Upgrading PostgreSQL versions and applying patches can be complex and risky.

***Problem Solving:*** *Plan upgrades carefully, test them in a staging environment, and follow PostgreSQL documentation and best practices. Use tools like pg\_upgrade where possible.*

**Security:** Inadequate security measures can lead to data breaches and unauthorized access.

***Problem Solving:*** *Implement proper authentication and authorization mechanisms, keep PostgreSQL and operating system patched, limit network access, and encrypt sensitive data.*

**Scaling Challenges:** Scaling PostgreSQL for high loads and large datasets can be challenging.

***Problem Solving:*** *Utilize techniques like connection pooling, read replicas, sharding, and vertical or horizontal scaling based on workload requirements.*

**28. Can you explain the role of checkpoints in PostgreSQL, and how do they impact database performance?  
Можете ли вы объяснить роль контрольных точек в PostgreSQL и как они влияют на производительность базы данных?** [**Source:**](https://www.postgresql.org/docs/current/wal-configuration.html#:~:text=Checkpoints%20are%20points%20in%20the,written%20to%20the%20WAL%20file.)

Checkpoints are points in the sequence of transactions at which it is guaranteed that the heap and index data files have been updated with all information written before that checkpoint, ensuring that committed changes are safely stored. At checkpoint time, all dirty data pages are flushed to disk and a special checkpoint record is written to the WAL file. In case of system failures, checkpoints facilitate recovery by enabling the system to reconstruct the database to a consistent state just before the failure

**29. How does PostgreSQL handle authentication and user management?  
Как PostgreSQL обрабатывает аутентификацию и управление пользователями?** [**Source:**](https://www.percona.com/blog/postgresql-database-security-what-you-need-to-know/#:~:text=PostgreSQL%20handles%20password%20security%20and,)%2C%20and%20encrypted%20password%20storage.)

PostgreSQL handles password security and authentication through features like strong password policies, support for various authentication methods (like LDAP, Kerberos, and certificate-based authentication), and encrypted password storage

**PostgreSQL provides various methods for authenticating users:**

* [Trust authentication](https://postgrespro.com/docs/postgresql/15/auth-trust), which simply trusts that users are who they say they are.
* [Password authentication](https://postgrespro.com/docs/postgresql/15/auth-password), which requires that users send a password.
* [GSSAPI authentication](https://postgrespro.com/docs/postgresql/15/gssapi-auth), which relies on a GSSAPI-compatible security library. Typically this is used to access an authentication server such as a Kerberos or Microsoft Active Directory server.
* [SSPI authentication](https://postgrespro.com/docs/postgresql/15/sspi-auth), which uses a Windows-specific protocol similar to GSSAPI.
* [Ident authentication](https://postgrespro.com/docs/postgresql/15/auth-ident), which relies on an “Identification Protocol” ([RFC 1413](https://tools.ietf.org/html/rfc1413)) service on the client's machine. (On local Unix-socket connections, this is treated as peer authentication.)
* [Peer authentication](https://postgrespro.com/docs/postgresql/15/auth-peer), which relies on operating system facilities to identify the process at the other end of a local connection. This is not supported for remote connections.
* [LDAP authentication](https://postgrespro.com/docs/postgresql/15/auth-ldap), which relies on an LDAP authentication server.
* [RADIUS authentication](https://postgrespro.com/docs/postgresql/15/auth-radius), which relies on a RADIUS authentication server.
* [Certificate authentication](https://postgrespro.com/docs/postgresql/15/auth-cert), which requires an SSL connection and authenticates users by checking the SSL certificate they send.
* [PAM authentication](https://postgrespro.com/docs/postgresql/15/auth-pam), which relies on a PAM (Pluggable Authentication Modules) library.
* [BSD authentication](https://postgrespro.com/docs/postgresql/15/auth-bsd), which relies on the BSD Authentication framework (currently available only on OpenBSD).

**30. What are the different types of joins supported by PostgreSQL, and when should each be used?  
Какие различные типы объединений поддерживаются PostgreSQL и когда следует использовать каждое из них?** [**Source:**](https://www.tutorialspoint.com/postgresql/postgresql_using_joins.htm)

**PostgreSQL - JOINS**

1. The CROSS JOIN.
2. The INNER JOIN.
3. The LEFT OUTER JOIN.
4. The RIGHT OUTER JOIN.
5. The FULL OUTER JOIN.

**Cross Join (or Cartesian Join):** Returns the Cartesian product of the two tables, meaning each row from the first table is combined with every row from the second table. Cross joins are used when you need to combine all rows from one table with all rows from another table, typically used with caution due to the potentially large result set.

**Inner Join:** Retrieves records that have matching values in both tables based on the specified join condition. Inner joins are used when you want to retrieve only the rows that have matching values in both tables.

**Left Outer Join (or Left Join):** Retrieves all records from the left table and the matched records from the right table. If there is no match, NULL values are returned for the columns from the right table. Left joins are useful when you want to retrieve all records from the left table, regardless of whether there are matching records in the right table.

**Right Outer Join (or Right Join):** Similar to left join but retrieves all records from the right table and the matched records from the left table. If there is no match, NULL values are returned for the columns from the left table. Right joins are less commonly used compared to left joins but can be useful in specific scenarios.

**Full Outer Join (or Full Join):** Retrieves all records from both tables and matches them where possible. If there is no match, NULL values are returned for the columns from the table without a matching row. Full joins are used when you want to retrieve all records from both tables, combining the results of left and right joins.