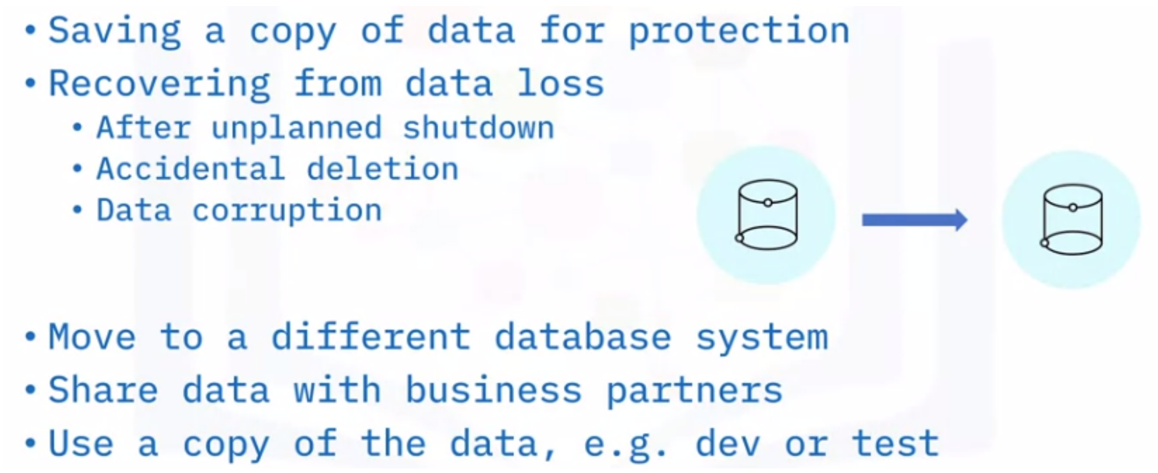
Week 2 - Managing Databases

Tipo	Anotaciones
Posición	
Etiquetas	
Bibliografía	
Fecha de creación	@June 12, 2022 7:30 AM
Property	

▼ Backup and Restore Databases

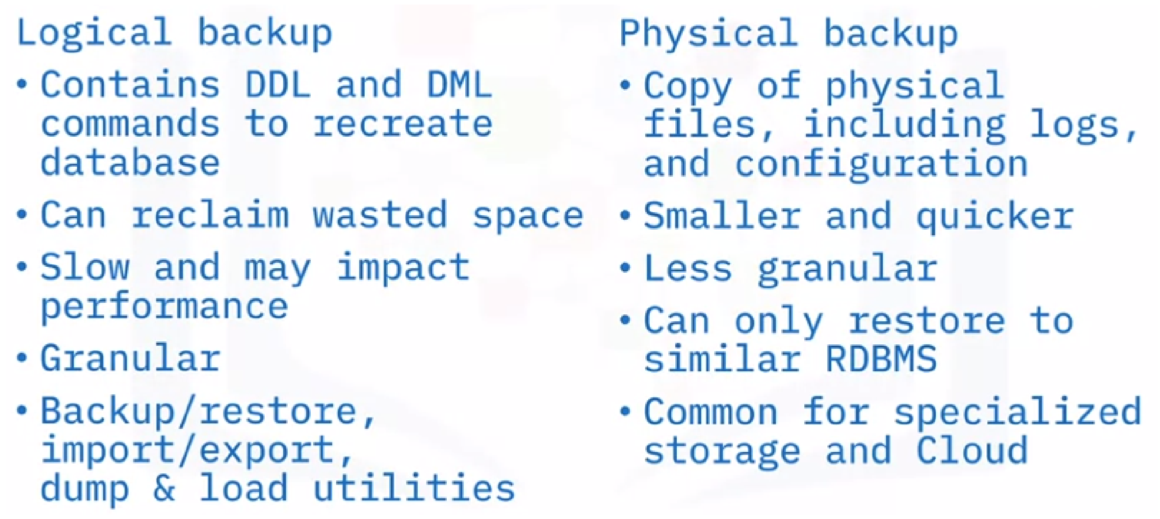
▼ Introduction to Backup and Restore

Backup and restore scenarios



- Saving a copy of data for protection
- Recovering from data loss
  - After unplanned shutdown
  - Accidental deletion
  - Data corruption
- Move to a different database system
- Share data with business partners
- Use a copy of the data, e.g. dev or test

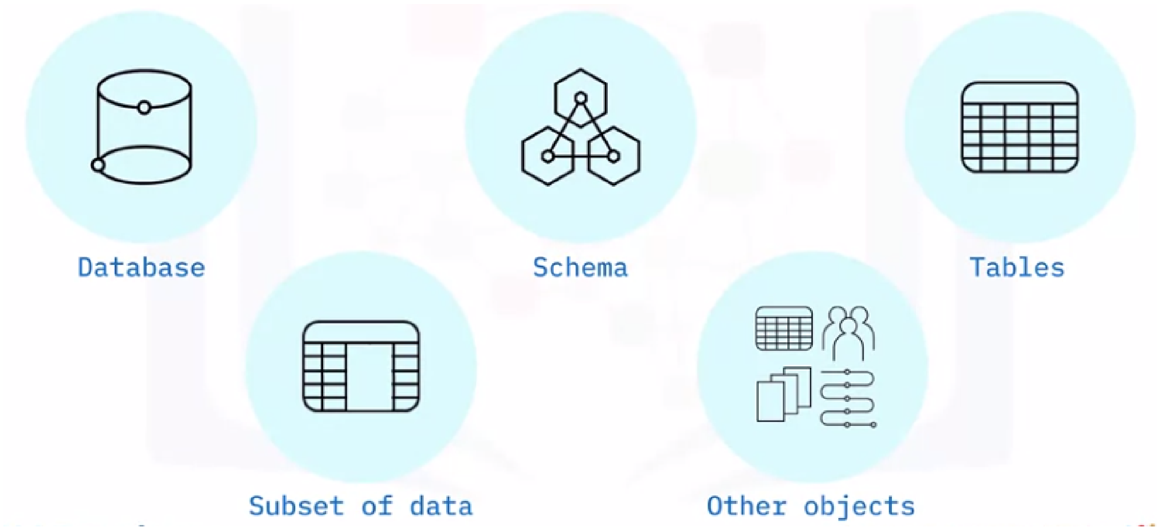
Logical backup vs Physical backup



Logical backup	Physical backup
• Contains DDL and DML commands to recreate database	• Copy of physical files, including logs, and configuration
• Can reclaim wasted space	• Smaller and quicker
• Slow and may impact performance	• Less granular
• Granular	• Can only restore to similar RDBMS
• Backup/restore, import/export, dump & load utilities	• Common for specialized storage and Cloud

When backing up databases, you can perform either logical or physical backups. A logical backup creates a file containing DDL (such as create table) and DML commands (such as insert) that recreate the objects and data in the database. As such, you can use this file to recreate the database on the same or another system. Generally, when you perform a logical backup and restore, you reclaim any wasted space from the original database because the restore process creates a clean version of the tables. Generating logical backups can take a long time for large databases and may impact the performance of other queries that are concurrently running. Logical backups enable you to backup granular objects. For example, you can back up an individual database or table; however, you cannot use it to backup log files or database configuration settings. You typically use import, export, dump, and load utilities to perform logical backups. A physical or raw backup creates a copy of all the physical storage files and directories that belong to a table, database, or other objects, including the data files, configuration files, and log files to aid point-in-time recovery. Physical backups are often smaller and quicker than logical backups; they are useful for large or important databases that require fast recovery times. They are similar to backing up any other types of files on your physical system; however, this means you won't be able to easily recover individual tables or database objects if a physical file contains data for more than one object. Because the backup file is specific to the RDBMS, you can only restore it to a similar system. This approach of taking storage level snapshots is common for databases utilizing specialized storage systems and on Cloud.

What to back up



You can choose exactly which parts of a database you want to backup. Depending on the RDBMS you are using and type of backup you are performing, you can back up

- a whole database
- the contents of a schema
- one or more tables from a database
- a subset of data from one or more tables in a database
- or a collection of other objects in the database.

Because you can also choose the regularity and type of backup you perform, you can customize your backup policies to exactly meet your needs. When using backup and restore, you should remember to check that your back up file or files are valid and that you can use your restore plan to restore them successfully. It is essential to check these when using backup and restore as part of your disaster recovery plans, as an invalid back up or an inability to restore can result in data loss. You should also ensure that you secure the transfer and storage location of your back up files at the same level that you secure the data in your database.

Backup options

- Compression:

    - Reduces size for storage and transmission
    - Increases time for backup and restore processes
- Encryption:
    - Reduces tha risk of data being compromised
    - Increases time for backup and restore processes

▼ Types of Backup

Types of backup

- Full backup
- Point-in-time backup
- Differential backup
- Incremental backup

Full backups

• Backs up all the specified data

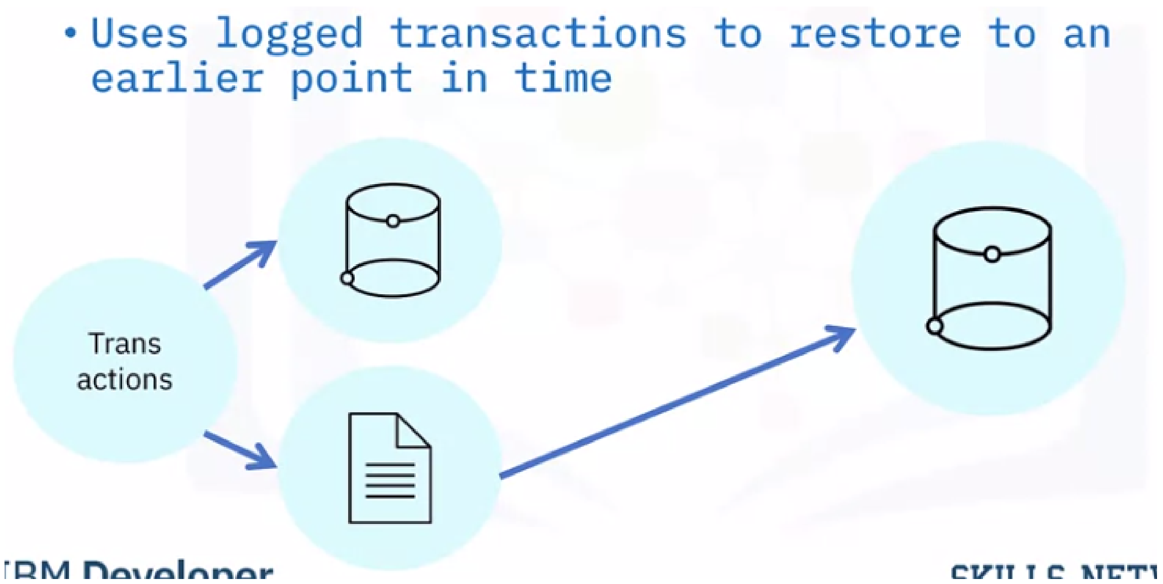
A full backup is a complete copy of all of the data in the object or objects that you are backing up. Performing full backups simplifies the restore and recovery processes, because you just need to locate the latest full backup and restore that one file. However, as the size of your database increases, the time, bandwidth, and storage for the backup file also increases.

- Multiple copies of the backup means storing many instances of a large file
- Only storing one copy risks data loss if file is corrupt

- Could be needlessly backing up unchanging data
- Must secure backup files

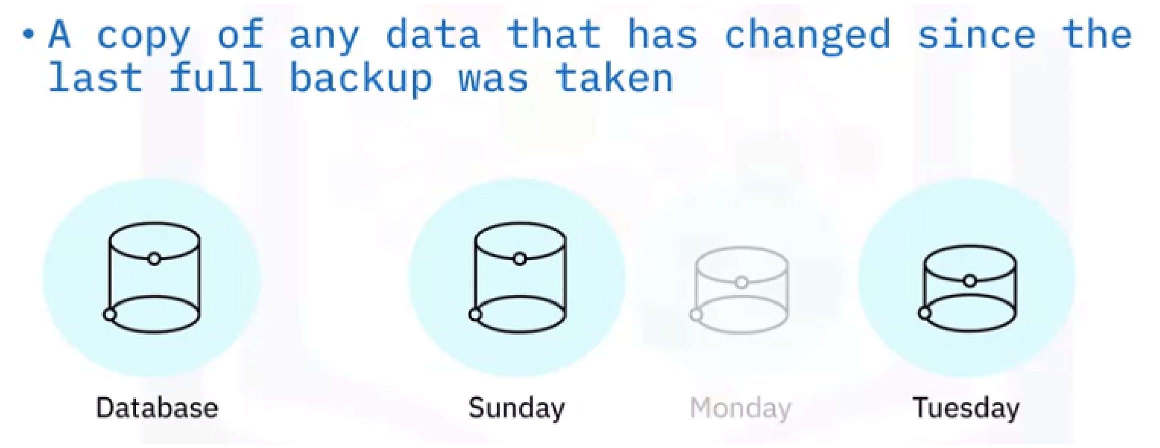
Point-in-time recovery

When you restore a full backup, you restore the data to the state that it was in when the backup was taken. But the database may have processed many transactions since that time, which should ideally be restored too. One way around this issue is to enable logging of each transaction on your database and then you can use the information in the log file to reapply the transactions to the restore database. The process of reapplying transactions after restoring a database backup is known as recovery and it enables you to recover the data to a state that it was in at a particular point in time, hence the name, point-in-time recovery.


• Uses logged transactions to restore to an earlier point in time
Trans actions

For example, if you know that a DML statement run at 11:05am erroneously deleted some data, you can restore the latest full backup and then reapply the transactions up to that point in time, minimizing the loss of data changes that occurred between the last full backup and the moment that the wrong data was deleted. Different database systems use different terminology for the log containing the transaction information, for example, MySQL calls it the binary log, Postgres calls it the write-ahead log, and Db2 on Cloud calls it the transaction log.

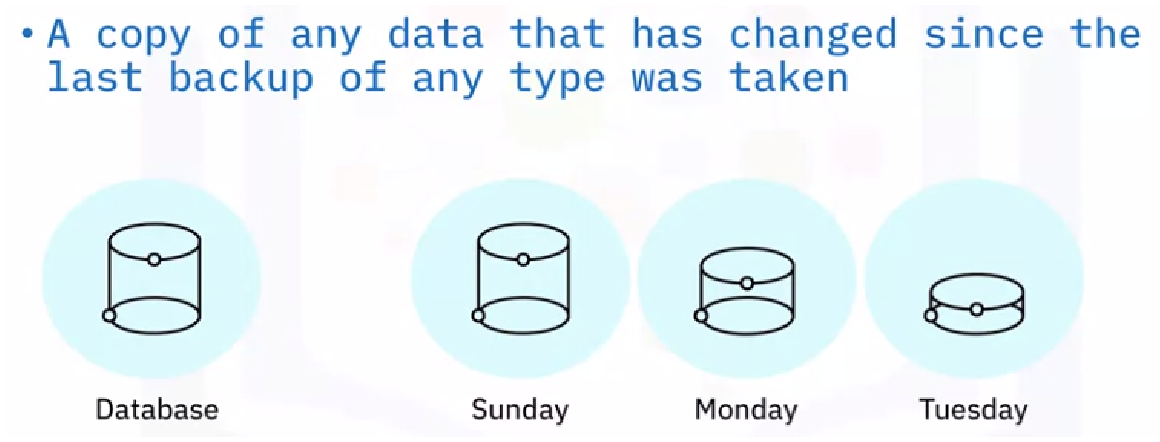
Differential backups


- A copy of any data that has changed since the last full backup was taken

Because full backups can require a lot of time, bandwidth, and storage, one alternative is to use them in conjunction with differential backups. A differential backup consists of a copy of any data that has changed since the last full backup was taken.

→ This makes the differential backup file much smaller than a full backup file, reducing the time, bandwidth, and storage needs for the backup while still enabling you to restore a recent copy of the data.

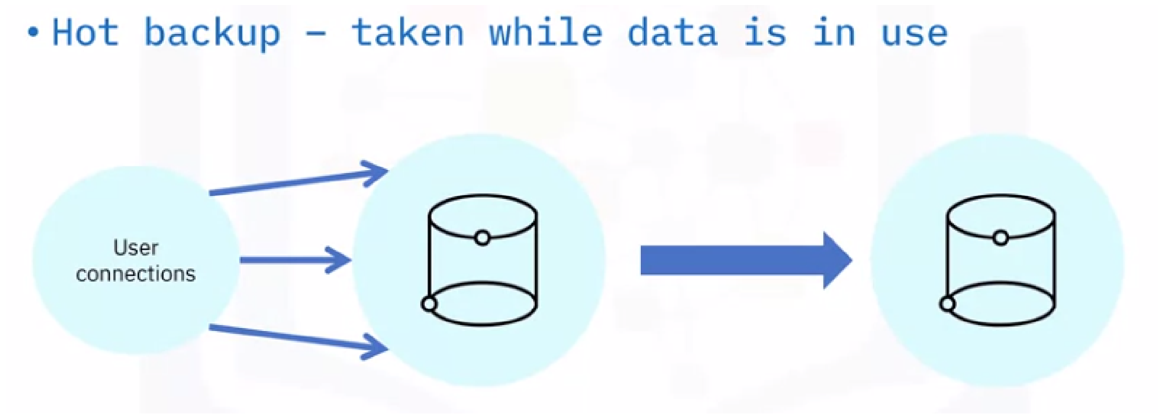
Incremental backup


- A copy of any data that has changed since the last backup of any type was taken

Incremental backups are similar to differential backups, but they only contain data that has changed since the last backup of any type was taken. Each incremental backup only contains changes since the backup was taken on the previous day, so if you need to restore the database on a Tuesday, you first restore the full backup from Sunday, then the incremental backup from Monday, and then the incremental backup from Tuesday.

▼ Backup Policies

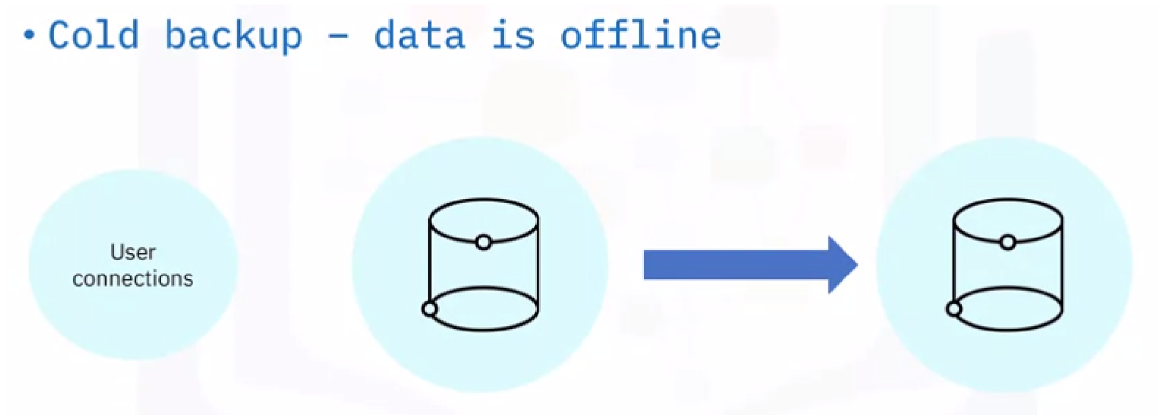
Hot vs. Cold backups



- Hot backup – taken while data is in use

- They have no impact on availability and users can continue with their activities throughout the backup period.
- However, hot backups can result in performance degradation for users while the backup is running and can impact on data integrity if data changes during the backup process.

Hot backups are generally stored on an available server and often receive regular updates from the production database. This enables them to be brought online should the production server fail, to ensure continuing availability to users.



- Cold backup – data is offline

- This eliminates the data integrity risks associated with hot backups, but has a greater impact on user availability and cannot be used in 24/7 environments.

Cold backups tend to be stored on external drives or on servers that are shut down between back up operations. This can provide greater data safety, but does mean that the recovery process will take longer than that of a hot backup.

Backup policies

- Physical or logical
- Full, differential, or incremental
- Hot or cold
- Compression
- Encryption
- Frequency:
    - Is data regularly changing or being added?
    - Is the existing table large?

  You also need to consider how often you need to back up your data. This is dependent on the impact that any data loss would have on your business operations and how frequently your data is changing. For example, a table containing product information is likely to change less frequently than a table containing customer orders, so you could back that up less regularly than the orders table. But if your orders table already contains a large amount of data, you won't want to be performing frequent full backups, so should consider using differential or incremental backups for it.
- Schedule:
    - Is the data accessed equally across the 24-hour day?
    - Is it accessed at weekends?

  You also need to consider when to perform your backups. If your data is mainly accessed during the working day in one time zone, you should schedule your backup outside of those hours. However, if it is accessed across the whole day, you could consider implementing full backups at a weekend and incremental or differential backups on a daily basis. Because backups will be a regular task, if your RDBMS provides the ability to schedule or automate your backups, you should consider using it.

- Automated

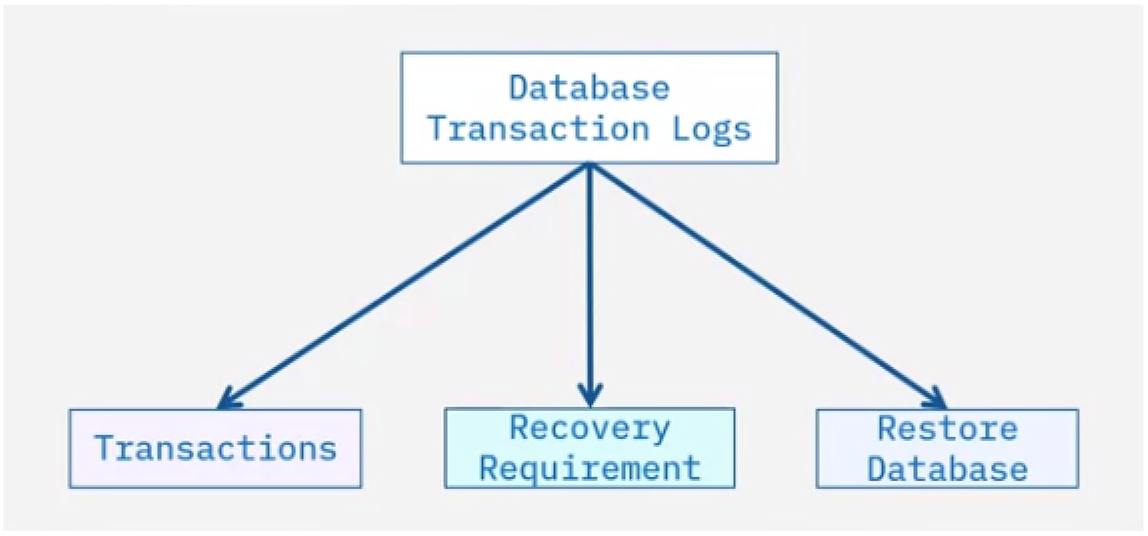
Managed cloud backups

Options dependant upon RDBMS and cloud service provider include:

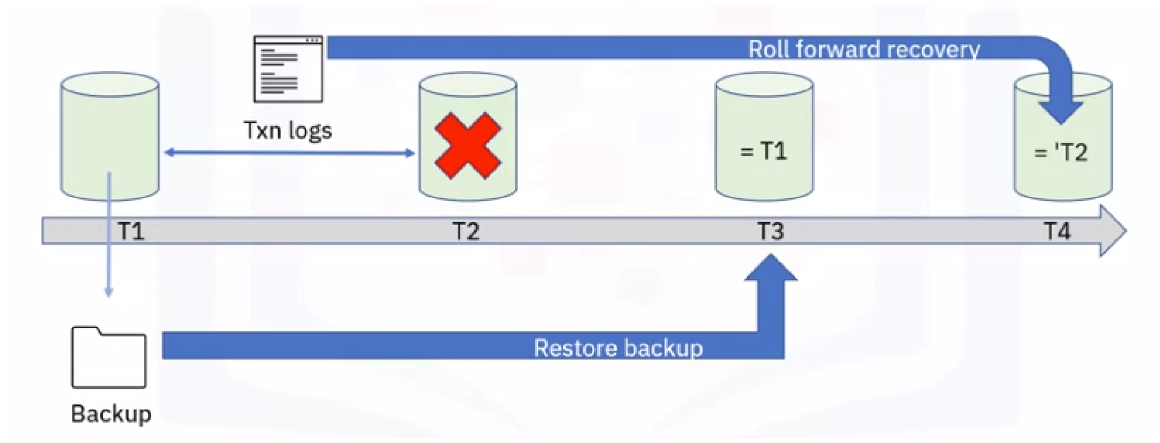
- Preconfigured automated backup
- Configurable automated backup
- Manual backup
- Third party tools

▼ Using Database Transaction Logs for Recovery

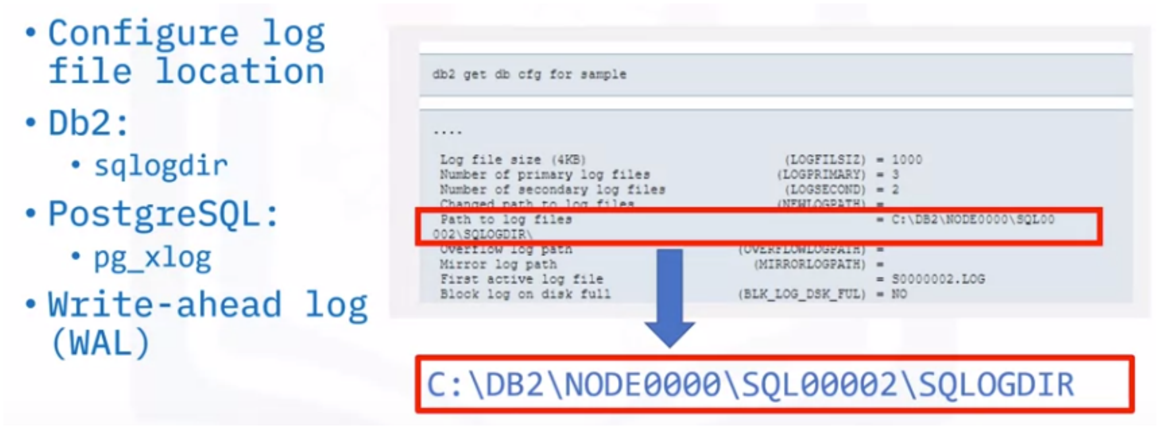
Database transaction logs



Example



Storing transaction log files



- Configure log file location
- Db2:
  - sqlogdir
- PostgreSQL:
  - pg_xlog
- Write-ahead log (WAL)

```
db2 get db cfg for sample
```

```
....

 Log file size (4KB)                          (LOGFILSIZ) = 1000
 Number of primary log files                 (LOGPRIMARY) = 3
 Number of secondary log files                (LOGSECOND) = 2
 Changed path to log files                   (NEWLOGPATH) =
 Path to log files                                        = C:\DB2\NODE0000\SQL00
002\SQLOGDIR\
 Overflow log path                      (OVERFLOWLOGPATH) =
 Mirror log path                          (MIRRORLOGPATH) =
 First active log file                                    = S0000002.LOG
 Block log on disk full                (BLK_LOG_DSK_FUL) = NO
```

```
C:\DB2\NODE0000\SQL00002\SQLOGDIR
```

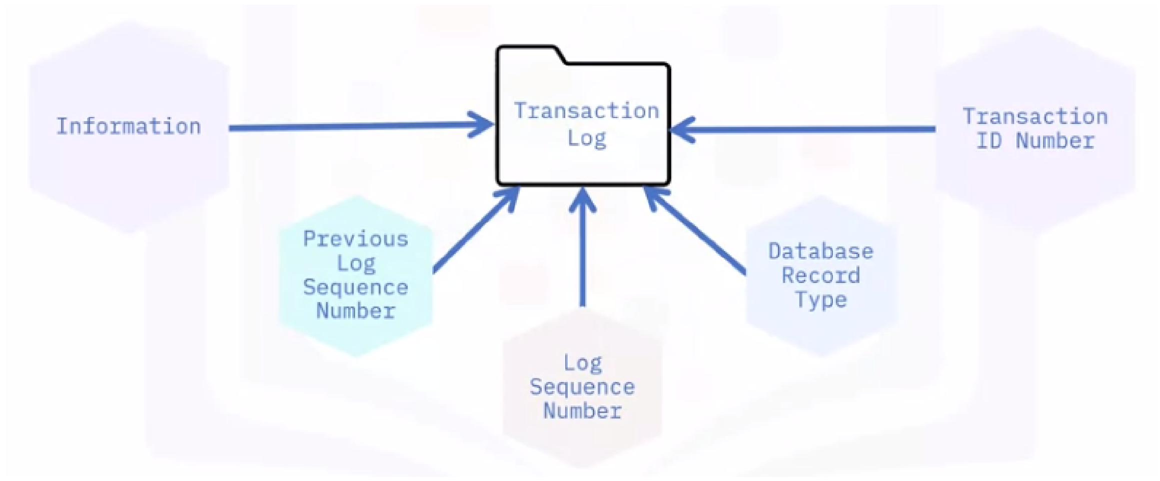
A recommended best practice:

- Isolate log files on storage volumes that are separate from where the database objects like tablespaces are stored.
  - Better **performance**, so database writes are not competing with log writes.
  - Improves **recoverability** since it isolates logs from crashes or the corruption of data volumes.
- **Log mirroring** - store a second copy of log files in an alternative location
- **Log shipping** - you can write scripts to copy or ship logs to remote replicas or standby systems

Accessing transaction log files

- Logging may or may not be enable by default.
    - - Log settings are usually configurable.
- MySQL - To view transaction log files:
    - SHOW BINARY LOGS;
- Transaction logs:
    - Tipically in binary format
    - May be encrypted
    - View using special tools
        - MySQL: mysqlbinlog -v DB_Bin_Logs.000001
        - Db2: db2fmtlog S0000002.LOG -replayonlywindow

Basic Anatomy of a database log



- Log Transaction ID Number - Unique ID for the log record.
- Database Record Type - Describes the type of database log record.
- Log Sequence Number - Reference to the database transaction generating the log record.
- Previous Log Sequence Number - Link to the last log record. This implies that database logs are constructed in linked list form.

- Information which details the actual changes that triggered the log record to be written.

🛠️ Hands-on Lab: Upload and Export using Db2 on Cloud
Ungraded External Tool: Hands-on Lab: Backup and Restore using PostgreSQL
Ungraded External Tool: Hands-on Lab: Backup and Restore using MySQL

▼ Security and User Management

▼ Overview of Database Security

Server security

- On-premise servers:
    - Who has access?
    - How are they physically secured?
- Managed cloud:
    - Check provider documentation

Operating system configuration

You should consider the operating system hosting your database.

- Regular patching
- System hardening
- Access monitoring

RDBMS configuration

- Regular patching

- Review and use system-specific security features
- Reduce the number of administrators

Accessing databases and objects

Users and client applications need to be permitted to access a server or database and the objects in it. Firstly, users need to be authenticated on the server or database to enable them to access it. And then they need to be granted permissions on individual objects, or groups of objects, in the database to interact with them.

Authentication

Database authentication is similar to the authentication you use when using a PIN or fingerprint to access your cell phone or a password to access your computer. It is a process of verifying that the user is who they claim to be, for example, by validating credentials such as username and password.

Some database systems enable you to use operating system or other credentials to authenticate against a database. For example, you can use external authentication methods, such as pluggable authentication module (PAM), Windows login IDs, lightweight directory access protocol (LDAP) or Kerberos.

- Authorized to:
    - Objects
    - Data
- Grant privileges to:
    - Users
    - Groups
    - Roles

Privileges

You can allow users to select, insert, update, or delete data, or to alter the structure of the table. In some RDBMSs, you can narrow this down even further to assign privileges on a columnar basis.

Auditing

- Monitor:
  - Who accesses what objects
  - What actions they perform
- Audit:
  - Actual access against security plan

Encryption

- Adds another layer of security:
  - Intruders need to decrypt
- Industry & regional regulations:
  - Algorithms
  - Key management
- Performance impact

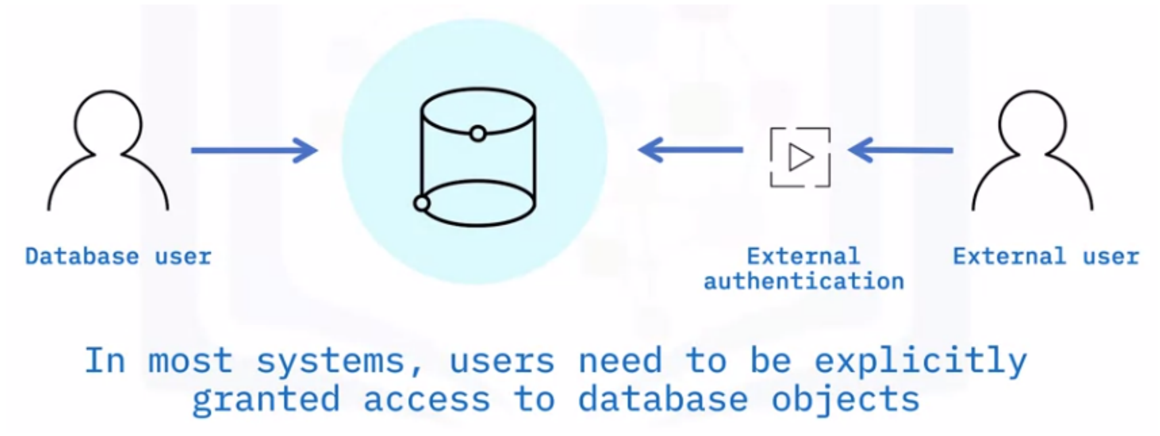
Application security

- SQL injection strings
- Insecure code

▼ Users, Groups, and Roles

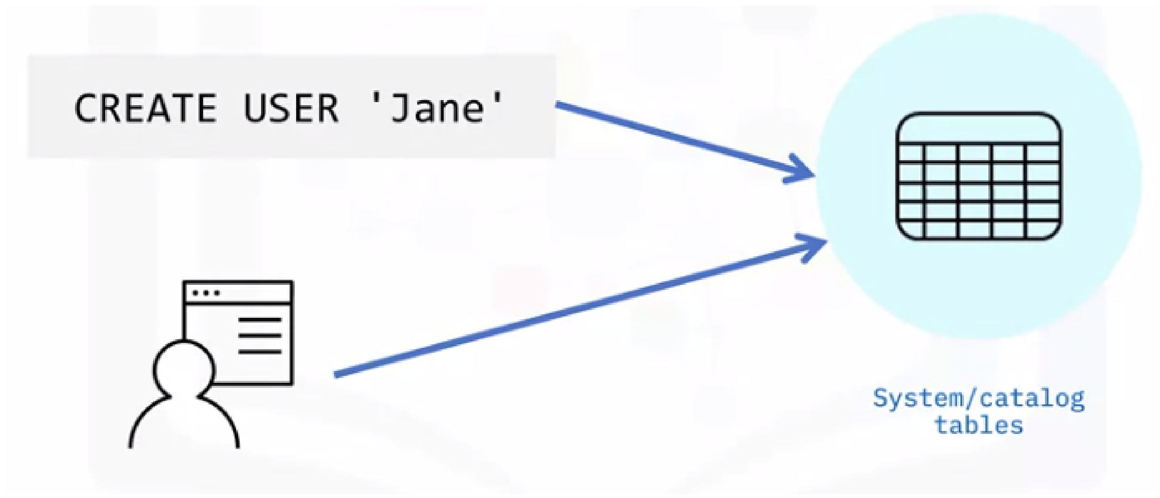
Database users

A database user is a user account that is allowed to access specified database objects. Depending on the DBMS and your security policies, a user might be explicitly created and authenticated within the database system, or may be created externally and authenticated using external authentication such as the operating system or external identity management services like Kerberos, LDAP, and Cloud IAM.



In most systems, users need to be explicitly granted access to database objects

User names are stored in system tables or catalog tables which you should not try to edit directly. All RDBMSs will provide SQL commands and/or user interface tools that you can use to manage your users.



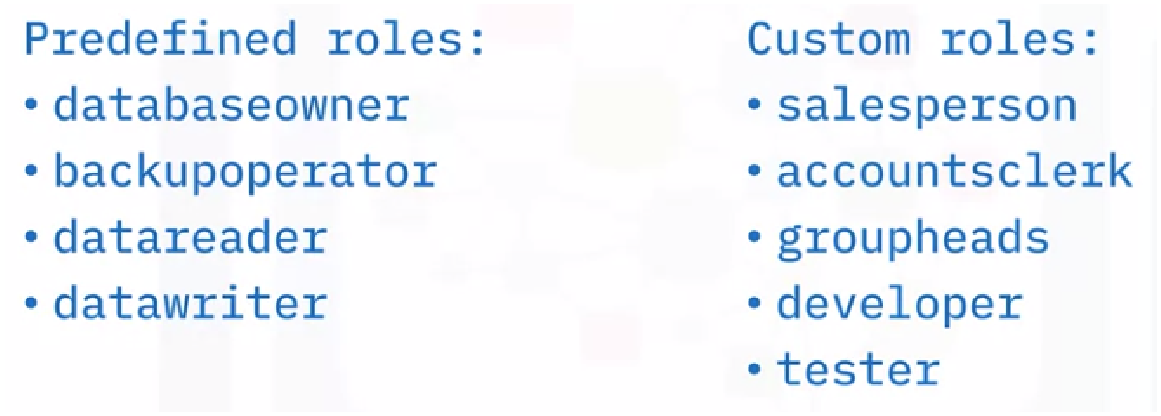
```
CREATE USER 'Jane'
```

Database groups

Some RDBMSs support the concept of user groups. In some cases, such as Postgres, you define groups in the database and they are logical groupings of users to simplify user management. In other systems, such as SQL Server and Db2, you can map a database group to an administrative group in the underlying operating system. This is particularly useful when you are using that operating system to authenticate your users. Similarly, if you are running your database on Amazon Relational Database Service (RDS) for example, you can use virtual private cloud (VPC) security groups and database (DB) security groups to manage your user access.

Database roles

Database roles are similar to database groups in that they confer privileges and access rights to all users of that role. A database role defines a set of permissions needed to undertake a specific role in the database. For example, a backup operator role would have permissions to access a database and to perform backup functions. Some RDBMSs have a set of predefined roles that you can use for your users, such as a database owner or backup operator role. And most enable you to create your own roles, so you might create a salesperson role that has the permissions a salesperson will need on the relevant tables in the database.



Predefined roles:

- databaseowner
- backupoperator
- datareader
- datawriter

Custom roles:

- salesperson
- accountsclerk
- groupheads
- developer
- tester

Managing security objects

Assigning privileges to groups or roles, rather than individual users, greatly simplifies your security management tasks.

- If you know that a set of users all need access to the same functionality to fulfill their job role, you can put those users in one group and assign the relevant permissions to the group. If that job role changes in the future, it is quicker, easier, and less prone to mistakes to just add the new permissions to the group rather than individual users.
- Similarly, if a new member of staff joins the team, you can simply add them to the role or group, rather than having to assign all the separate permissions to them.

One user can be a member of one or more roles.

Principle of least privilege

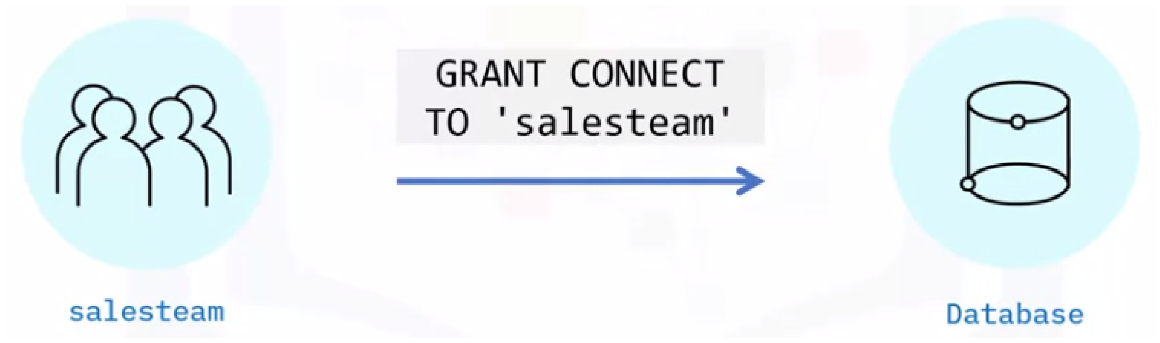
Only adding users to groups they need to be a member of, and ensuring that your roles don't include any permissions that the majority of the users in them will not need. In the latter scenario, you should reassess your role permissions and create multiple, more granular, roles or groups.

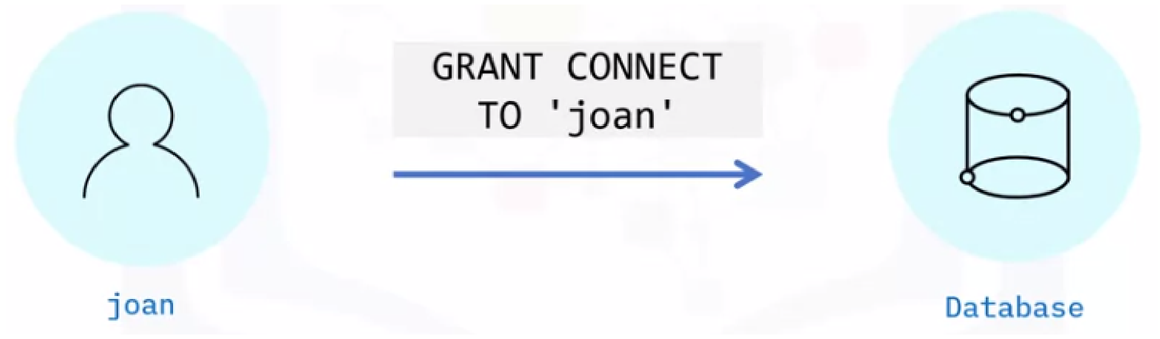
▼ Managing Access to Databases and Their Objects

Authorization

After a user is authenticated on a database, they need permissions or privileges to access the objects and data in that database. Privileges are granted to users, and to groups or roles. The combination of a user's own personal permissions and those of the groups or roles they belong to defines their overall permissions.

Database access





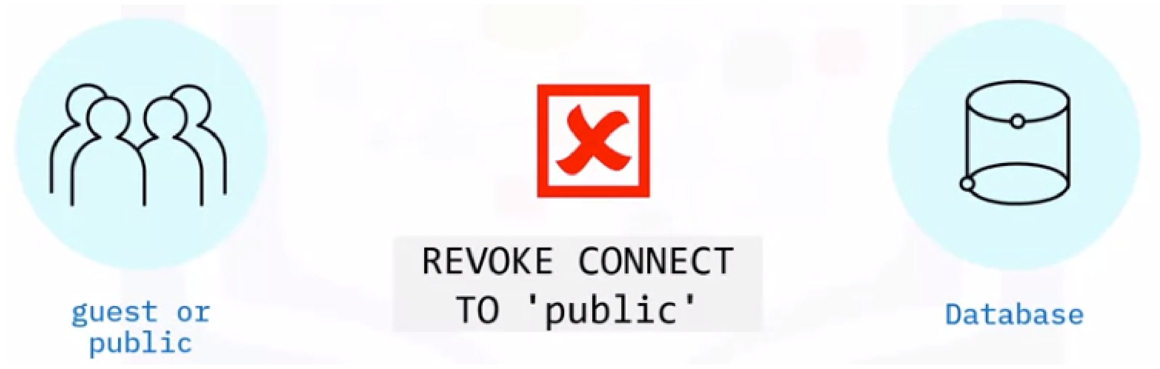
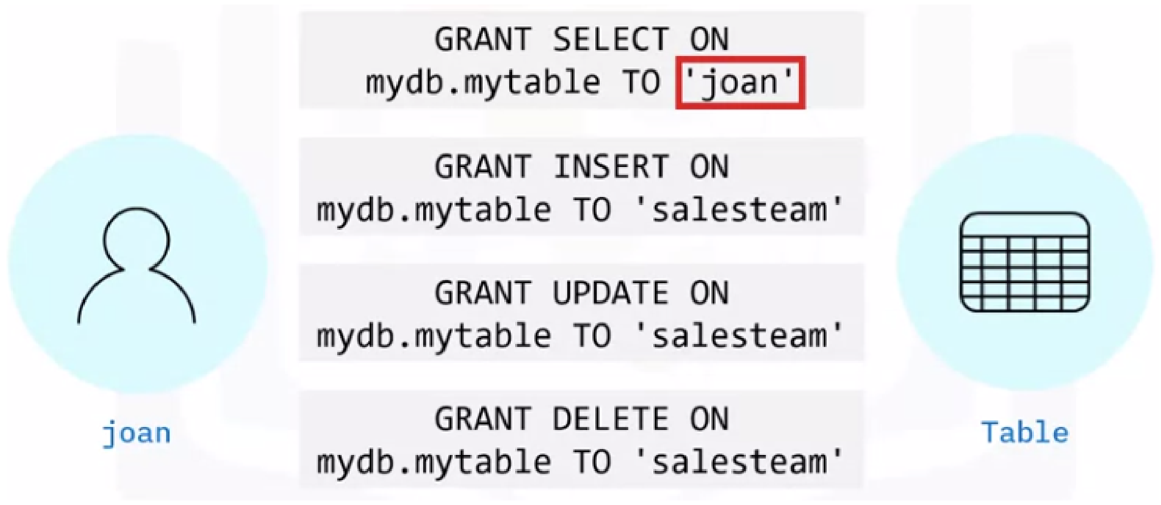
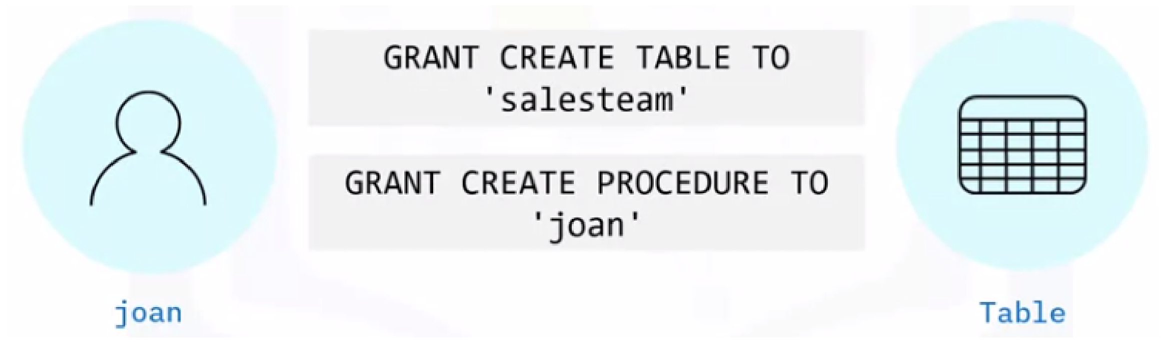


Table access

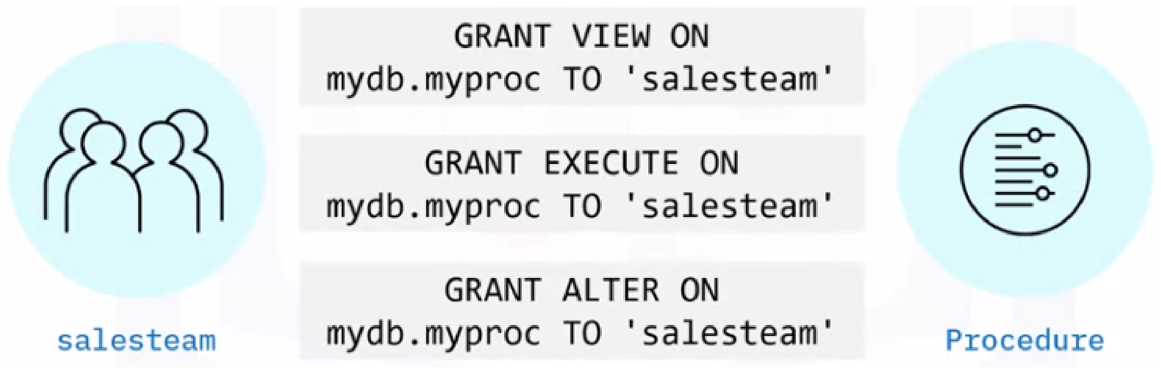


```
GRANT SELECT ON
mydb.mytable TO 'joan'

GRANT INSERT ON
mydb.mytable TO 'salesteam'

GRANT UPDATE ON
mydb.mytable TO 'salesteam'

GRANT DELETE ON
mydb.mytable TO 'salesteam'
```

Object definition access



```
GRANT CREATE TABLE TO
'salesteam'

GRANT CREATE PROCEDURE TO
'joan'
```



```
GRANT VIEW ON
mydb.myproc TO 'salesteam'

GRANT EXECUTE ON
mydb.myproc TO 'salesteam'

GRANT ALTER ON
mydb.myproc TO 'salesteam'
```

On a procedure or function, you can permit a user or group to view the code. This means that they can view the definition of that function or procedure, but they will not be able to run it or change it. To run the code, a user needs the execute permission. And to change the definition of a procedure, a user needs the alter permission.

Revoke and deny access

```
REVOKE SELECT ON mydb.mytable TO 'salesteam'
```

```
DENY SELECT ON mydb.mytable TO 'salesteam'
```

You can use the revoke statement to remove granted privileges, and most RDBMSs will also provide revoke functionality in the user interface. However, because an individual’s privileges are a combination of those granted to all the groups or roles they belong to, a member of the sales team role may still be able to access this table through their membership of another role. If you want to ensure that users do not have permission for a certain object or action, you can use the deny statement to override any previous grant of that permission.

▼ Auditing Database Activity

Why audit your database?

- Does not directly offer protection
- But it does help you to:
  - Identify gaps in security system

    - Track errors in privilege administration
- Compliance
- You should consider implementing auditing in all your database solutions, whether they be on premise or in the cloud.

Auditing a database involves recording the access and the activity of database users on your database objects. By reviewing such records, also known as **logs**, you can identify suspicious activity and quickly respond to any security threats you find.

Auditing database access

It is imperative that you track who accesses your database and review the information to identify any unauthorized users. You should also track failed attempts to access the database, as these can help you to identify potential attacks, such as brute force attempts, on your system.

- Db2 on Cloud - built-in audit factility to log user authentification events
- MySQL - audit log plogin tracks connect and disconnect events

Auditing database activity

To audit database activity, some RDBMSs use triggers. These are special stored procedures that automatically log the activity after a DML statement event, such as an insert, occurs. Other systems enable you to attach actions to the events that occur in the database.

- Db2 on Cloud - change history event monitor
- PostgreSQL - pgAudit

You should ensure that whatever auditing implementation you use meets the compliance requirements of your customer or region.

▼ Encrypting Data

Why encrypt your data?

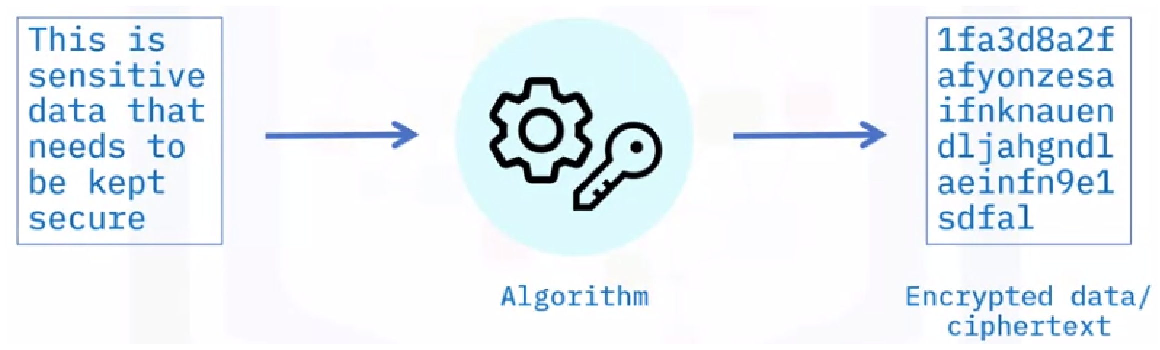
- Another layer in security system

- Often last line of defense
- Can protect data:
    - At rest
    - During transmission
- May be required by:
    - Industry
    - Region
    - Customer

Protecting data at rest

"At rest data" refers to your data when it is stored, or at rest, in your RDBMS. It is important to encrypt data at rest to ensure that malicious users cannot directly access the data files, bypassing your database security measures. Some RDBMSs provide database-level encryption for the entire database. This is often termed transparent data encryption, or TDE. Some provide table or tablespace encryption for individual tables. And some provide column level for maximum granularity and flexibility. Do note though, that the flexibility that column level encryption provides is counteracted by performance degradation and complexity of set up.

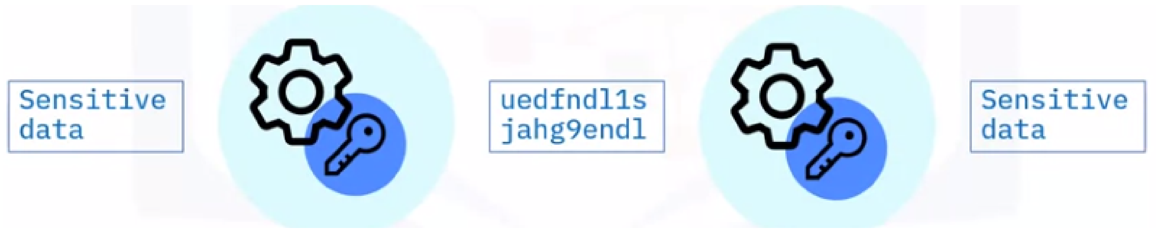
Algorithm and keys


This is sensitive data that needs to be kept secure
Algorithm
1fa3d8a2f afyonzesa ifnknauen dljahgndl aeinfn9e1 sdfal
Encrypted data/ ciphertext

You encrypt data by using an algorithm to change the data into an unreadable string, commonly known as ciphertext. These algorithms use a key to ensure that anyone trying to decipher the text cannot do so without the key.

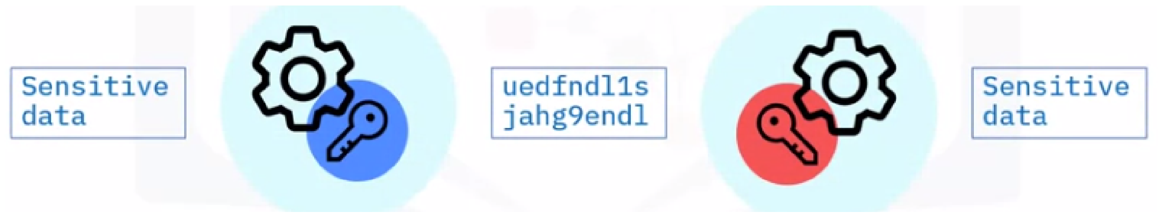
Symmetric encryption

- Same key to encrypt and decrypt
- Examples: Advanced Encryption Standard (AES) and Data Encryption Standard (DES)
- Key is shared with all users
- Increased likelihood of compromised



Asymmetric encryption

- Use two keys (key pair): one public, one private
- Public key encrypts, private key decrypts
- Rives-Shamir-Adleman (RSA) and elliptic curve cryptography (ECC)
- Must store private keys securely



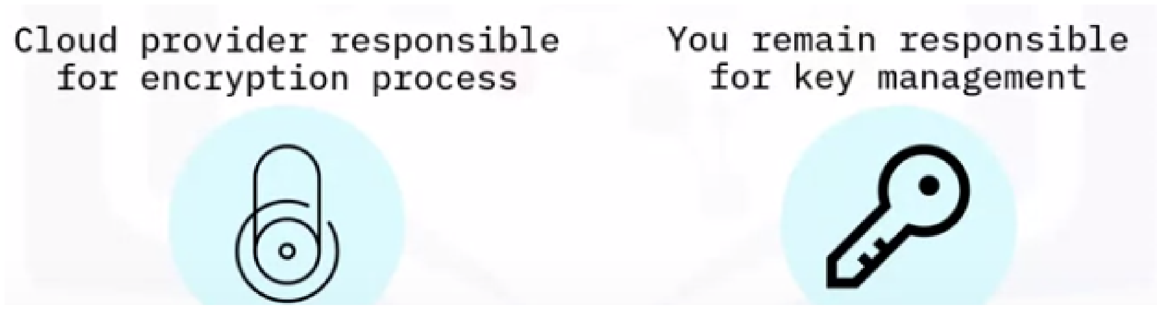
Transparent data encryption

Most modern databases support transparent data encryption, or TDE.

- The encryption and decryption is not visible to your users. The database engine encrypts the data before storing it and then decrypts it when an authorized user or application requests information.
- Because the database engine is performing the encryption tasks, it also encrypts your database backups too.

Customer managed keys

Although TDE simplifies the process of encrypting your database, you may have concerns about storing data in the cloud and relinquishing the key management process to a third party. Some cloud databases support customer managed keys, also known as the Bring Your Own Key (or BYOK) protocol.


Cloud provider responsible for encryption process
You remain responsible for key management

- Benefits:
  - Cloud provider cannot access your confidential data
  - Security admins manage keys; database admins manage data
  - Complete control over keys and their lifecycle

Encryption vs. Performance

- Choice of symmetric or asymmetric encryption may be configurable
- All encryption takes time and effort
- Asymmetric algorithms generally use longer keys, therefore have greater overheads
- Symmetric algorythms are often sufficient

Protecting data in transit

- Often provided by the RDBMS
- Protocols:
    - Transport Layer Security (TLS) protocol
    - Secure Sockets Layer (SSL)
- May encrypt by default, may be configurable
- Performance impact

🛠️ Reading: User Management with DB2
Ungraded External Tool: Hands-on Lab: MySQL User Management, Access Control, and Encryption
Ungraded External Tool: Hands-on Lab: User Management and Access Control in PostgrSQL