Backup y Restauración

Bases de Datos 1 (CC3088) - 2026

Backup y Restauración

Semestre 01, 2026

La importancia de los backups

Los datos se pierden.

No es una pregunta de si ocurrirá. Es una pregunta de cuándo.

Causas reales de pérdida de datos

| Causa | Frecuencia | | --- | --- | | Error humano (DELETE sin WHERE, DROP TABLE) | La más común | | Fallo de hardware (disco duro, servidor) | Frecuente | | Corrupción de datos | Ocasional | | Ransomware / ataque cibernético | En aumento | | Bug de software | Ocasional | | Desastre físico (inundación, incendio) | Raro pero total |

El error humano es responsable de la mayoría de las pérdidas de datos en producción.

El costo de no tener backup

Una empresa de e-commerce pierde su base de datos en viernes a las 11 PM.

- Sin backup: los datos de clientes, pedidos e inventario son irrecuperables.
- El negocio no puede operar hasta reconstruir — si es que puede.
- Pérdida económica directa + reputación + posibles problemas legales.

El costo de implementar backups es una fracción mínima del costo de no tenerlos.

La ilusión de seguridad

Muchos sistemas tienen backup en teoría.

En la práctica:

- El backup se configuró hace 2 años y nadie verificó si sigue funcionando.
- El backup existe pero el archivo está corrupto.
- El backup existe pero restaurar toma 6 horas que el negocio no puede permitirse.

Un backup que nunca se ha restaurado no es un backup. Es una esperanza.

Conceptos fundamentales

RPO — Recovery Point Objective

La cantidad máxima de datos que el sistema puede perder.

Si el backup es diario a las 2 AM y el fallo ocurre a las 11 PM, se pierden ~21 horas de datos.

El RPO define la frecuencia mínima del backup.

| Negocio | RPO aceptable | | --- | --- | | Blog personal | 24 horas | | E-commerce pequeño | 1-4 horas | | Sistema bancario | Segundos | | Sistema médico crítico |

Cero (replicación en tiempo real) |

RTO — Recovery Time Objective

El tiempo máximo que el sistema puede estar inactivo.

Un backup de 100 GB puede tardar 2 horas en restaurarse.

Si el RTO del negocio es 30 minutos, ese proceso no es suficientemente rápido.

El RTO define qué tipo de estrategia de recovery se necesita.

| Negocio | RTO aceptable | | --- | --- | | Blog personal | Horas / días | | E-commerce mediano | 1-2 horas | | Sistema bancario | Minutos |

Tipos de backup

| Tipo | Contenido | Tamaño | Velocidad | | --- | --- | --- | --- | | Full | Todo, siempre | Grande | Lenta | | Incremental | Solo lo que cambió desde el último backup (full o incremental) | Pequeño | Rápida | | Diferencial | Todo lo que cambió desde el último backup full | Mediano | Media |

Full vs Incremental en restauración

Full solo:

```
Restaurar = aplicar 1 backup
Riesgo    = solo 1 archivo a gestionar
```

Full + Incremental:

```
Restaurar = aplicar backup full + cada incremental en orden
Riesgo    = si falta un incremental, la cadena se rompe
```

Full + Diferencial:

```
Restaurar = aplicar backup full + el último diferencial
Balance   = menos archivos que incremental, más que full solo
```

La regla 3-2-1

La estrategia estándar de la industria para backups confiables.

3 copias de los datos.

2 tipos de medios de almacenamiento diferentes.

1 copia offsite (fuera de las instalaciones físicas).

Por qué cada número importa

3 copias: la copia en producción + 2 backups. Si una falla, hay dos más.

2 medios distintos: no guardar todo en el mismo tipo de almacenamiento. Un fallo de tipo (ej: todos los discos del mismo fabricante con el mismo defecto) no elimina todo.

1 offsite: un desastre físico (incendio, inundación) destruye todo lo que está en el mismo lugar. La copia offsite sobrevive.

Implementación práctica

```
Copia 1: Base de datos en producción (servidor principal)
Copia 2: Backup en servidor de backup local (NAS, otro servidor)
Copia 3: Backup en almacenamiento en la nube (S3, Google Cloud Storage)

Medio 1: Disco del servidor
Medio 2: Almacenamiento en la nube

Offsite: La copia en la nube está en una ubicación geográfica diferente
```

La regla 3-2-1-1-0 (versión moderna)

Extensión recomendada en entornos críticos:

- +1 copia inmutable (no puede ser modificada ni eliminada — protección contra ransomware).
- +0 errores al verificar la restauración — ningún backup no verificado cuenta.

PostgreSQL: pg_dump

`pg_dump` es la herramienta estándar para hacer backup de una base de datos PostgreSQL.

Genera un script SQL o un archivo binario que puede restaurarse en cualquier instalación compatible.

Backup completo en formato SQL

```
pg_dump -U usuario -h localhost -d nombre_db > backup.sql
```

El archivo resultante contiene todos los `CREATE TABLE`, `INSERT`, índices, vistas y funciones.

Backup con compresión

```
# Formato custom (comprimido, permite restauración selectiva)
pg_dump -U usuario -h localhost -d nombre_db -Fc > backup.dump

# Formato directory (paralelizable)
pg_dump -U usuario -h localhost -d nombre_db -Fd -f backup_dir/
```

El formato custom (`-Fc`) es el recomendado para producción:

- Comprimido — ocupa menos espacio.
- Permite restaurar tablas individuales.
- Compatible con `pg_restore`.

Backup de todas las bases de datos

```
# pg_dumpall incluye roles, tablespaces y todas las bases de datos
pg_dumpall -U postgres -h localhost > backup_completo.sql
```

Usar cuando se necesita migrar o recuperar un servidor completo.

Opciones útiles de pg_dump

```
# Solo el esquema (sin datos)
pg_dump -U usuario -d nombre_db --schema-only > esquema.sql

# Solo los datos (sin CREATE TABLE)
pg_dump -U usuario -d nombre_db --data-only > datos.sql

# Solo una tabla específica
pg_dump -U usuario -d nombre_db -t nombre_tabla > tabla.sql

# Excluir una tabla
pg_dump -U usuario -d nombre_db --exclude-table=logs > backup_sin_logs.sql
```

PostgreSQL: Restauración

Restaurar desde SQL plano

```
# Crear la base de datos primero (si no existe)
createdb -U postgres nueva_db

# Restaurar
```

```
psql -U usuario -d nueva_db < backup.sql
```

Restaurar desde formato custom con pg_restore

```
# Restaurar completo
pg_restore -U usuario -d nombre_db -Fc backup.dump

# Restaurar solo una tabla específica
pg_restore -U usuario -d nombre_db -Fc -t productos backup.dump

# Restaurar con verbose — ver el progreso
pg_restore -U usuario -d nombre_db -Fc -v backup.dump

# Solo el esquema
pg_restore -U usuario -d nombre_db -Fc --schema-only backup.dump
```

Restaurar en paralelo (más rápido para bases grandes)

```
# -j 4 usa 4 workers en paralelo — útil para backups grandes
pg_restore -U usuario -d nombre_db -Fd -j 4 backup_dir/
```

MySQL: mysqldump

`mysqldump` es la herramienta estándar para backup en MySQL.

Genera un script SQL con toda la estructura y los datos.

Backup completo

```
mysqldump -u usuario -p nombre_db > backup.sql
```

Backup con todas las bases de datos

```
mysqldump -u root -p --all-databases > backup_completo.sql
```

Opciones importantes

```
# Incluir rutinas (stored procedures, funciones)
mysqldump -u usuario -p --routines nombre_db > backup.sql

# Incluir triggers
mysqldump -u usuario -p --triggers nombre_db > backup.sql

# Solo esquema
mysqldump -u usuario -p --no-data nombre_db > esquema.sql

# Solo datos
mysqldump -u usuario -p --no-create-info nombre_db > datos.sql

# Transacción consistente (para InnoDB — snapshot sin bloquear)
mysqldump -u usuario -p --single-transaction nombre_db > backup.sql

# Con compresión
mysqldump -u usuario -p nombre_db | gzip > backup.sql.gz
```

`--single-transaction` es esencial en producción para no bloquear la base de datos durante el backup.

Restaurar en MySQL

```
# Restaurar en una base de datos existente
mysql -u usuario -p nombre_db < backup.sql

# Restaurar desde archivo comprimido
gunzip -c backup.sql.gz | mysql -u usuario -p nombre_db

# Crear la base de datos y restaurar
mysql -u root -p -e "CREATE DATABASE nueva_db;"
```

```
mysql -u root -p nueva_db < backup.sql
```

Un backup no es un backup hasta que se restaura

Esta es la parte que casi todos omiten.

Un archivo de backup puede estar:

- Corrupto — el disco tuvo un error al escribir.
- Incompleto — el proceso se interrumpió.
- Irrestaurable — versión incompatible, permisos incorrectos, dependencias faltantes.

El único test válido es: ¿se puede restaurar y funciona?

Prueba de restauración: proceso

```
# 1. Crear una base de datos de prueba
createdb -U postgres test_restore

# 2. Restaurar el backup ahí
pg_restore -U postgres -d test_restore backup.dump

# 3. Verificar la integridad
psql -U postgres -d test_restore -c "SELECT COUNT(*) FROM clientes;"
psql -U postgres -d test_restore -c "SELECT COUNT(*) FROM pedidos;"
psql -U postgres -d test_restore -c "SELECT COUNT(*) FROM productos;"

# 4. Comparar con producción
# Los conteos deben coincidir (o estar dentro de lo esperado según cuándo se hizo el back

# 5. Limpiar
```

```
dropdb -U postgres test_restore
```

Frecuencia de las pruebas de restauración

| Criticidad del sistema | Frecuencia de prueba | | --- | --- | | Producción crítica | Semanal o con cada backup | | Producción estándar | Mensual | | Desarrollo / staging | Antes de cada migración importante |

Si no se puede probar con frecuencia, al menos documentar el proceso detallado para que sea reproducible bajo presión.

Automatización con scripts

Script de backup PostgreSQL

```
#!/bin/bash
# backup_postgres.sh

DB_NAME="mi_base_de_datos"
DB_USER="usuario"
BACKUP_DIR="/var/backups/postgres"
FECHA=$(date +%Y-%m-%d_%H-%M-%S)
ARCHIVO="$BACKUP_DIR/${DB_NAME}_${FECHA}.dump"
RETENTION_DAYS=30

# Crear directorio si no existe
mkdir -p "$BACKUP_DIR"

# Hacer el backup
pg_dump -U "$DB_USER" -Fc "$DB_NAME" > "$ARCHIVO"

# Verificar que el backup se generó correctamente
if [ $? -eq 0 ]; then
    echo "[$FECHA] Backup exitoso: $ARCHIVO"
else
```

```
    echo "[$FECHA] ERROR: El backup falló" >&2
    exit 1
fi

# Eliminar backups más antiguos que RETENTION_DAYS días
find "$BACKUP_DIR" -name "*.dump" -mtime +$RETENTION_DAYS -delete

echo "Backups eliminados con más de $RETENTION_DAYS días."
```

Script de backup MySQL

```
#!/bin/bash
# backup_mysql.sh

DB_NAME="mi_base_de_datos"
DB_USER="usuario"
DB_PASS="contraseña"
BACKUP_DIR="/var/backups/mysql"
FECHA=$(date +%Y-%m-%d_%H-%M-%S)
ARCHIVO="$BACKUP_DIR/${DB_NAME}_${FECHA}.sql.gz"
RETENTION_DAYS=30

mkdir -p "$BACKUP_DIR"

# Backup con transacción consistente + compresión
mysqldump -u "$DB_USER" -p"$DB_PASS" \
    --single-transaction \
    --routines \
    --triggers \
    "$DB_NAME" | gzip > "$ARCHIVO"

if [ $? -eq 0 ]; then
    echo "[$FECHA] Backup exitoso: $ARCHIVO"
else
    echo "[$FECHA] ERROR: El backup falló" >&2
    exit 1
fi
```

```
find "$BACKUP_DIR" -name "*.sql.gz" -mtime +$RETENTION_DAYS -delete
```

Automatizar con cron

```
# Editar el crontab
crontab -e

# Backup diario a las 2:00 AM
0 2 * * * /usr/local/bin/backup_postgres.sh >> /var/log/backup.log 2>&1

# Backup cada 6 horas (para sistemas de mayor criticidad)
0 */6 * * * /usr/local/bin/backup_postgres.sh >> /var/log/backup.log 2>&1

# Backup semanal completo los domingos a las 1:00 AM
0 1 * * 0 /usr/local/bin/backup_postgres_full.sh >> /var/log/backup.log 2>&1
```

El `>> /var/log/backup.log 2>&1` redirige tanto stdout como stderr al log.

Enviar el backup a la nube

```
#!/bin/bash
# Después de generar el backup local, copiar a S3

ARCHIVO="$1"
BUCKET="s3://mi-empresa-backups/postgres/"

# Requiere AWS CLI configurado
aws s3 cp "$ARCHIVO" "$BUCKET"

if [ $? -eq 0 ]; then
    echo "Backup subido a S3: $BUCKET"
else
    echo "ERROR: No se pudo subir a S3" >&2
fi
```

Script de verificación automática

```
#!/bin/bash
# verificar_backup.sh — restaurar y contar filas

BACKUP="$1"
TEST_DB="test_restore_$(date +%s)"

createdb -U postgres "$TEST_DB"

pg_restore -U postgres -d "$TEST_DB" "$BACKUP"

if [ $? -ne 0 ]; then
    echo "ERROR: La restauración falló en $BACKUP" >&2
    dropdb -U postgres "$TEST_DB"
    exit 1
fi

# Verificar tablas críticas
CLIENTES=$(psql -U postgres -d "$TEST_DB" -tAc "SELECT COUNT(*) FROM clientes;")
PEDIDOS=$(psql -U postgres -d "$TEST_DB" -tAc "SELECT COUNT(*) FROM pedidos;")

echo "Verificación OK — Clientes: $CLIENTES | Pedidos: $PEDIDOS"

dropdb -U postgres "$TEST_DB"
```

Estrategia completa recomendada

| Frecuencia | Tipo | Destino | Retención | | --- | --- | --- | --- | | Cada 6 horas | Full (`pg_dump -Fc`) | Servidor local | 7 días | | Diario 2 AM | Full + verificación | Servidor local + S3 | 30 días | | Semanal domingo | Full | S3 (otra región) | 1 año | | Mensual | Full | Almacenamiento frío (Glacier) | 5 años |

El flujo completo

```
Producción → pg_dump → backup local (Copia 2)
                     ↓
```

```
                aws s3 cp → S3 (Copia 3, offsite)

Cada semana: restaurar en base de datos de prueba → verificar conteos → limpiar
```