Backup, Techniques and Strategies

- **Backup, validation and recovery of a database**
- **Types of backup and common options**
- **Design of a backup and recovery strategy**
- **Best practices**

Database backup

- A backup is a full or partial copy of the information in a database, held in a physically separate location.
- If the main database file becomes unavailable, you can restore a copy of it from the backup, and use the current transaction log to recover any additional data that may not be present in the database you restored.

Database validation

- Database file corruption may not be reported until the database server tries to access the affected part of the database. So it is important periodically to check that your database is valid.
- Depending on the options you specify, validation can include checksums, correctness of index data, and whether all table pages belong to objects in the database
- The Validation utility can be used in combination with regular backups to give you confidence in the integrity of the data in your database. To validate a backup copy of your database, make a copy of the backup and validate the copy. This process ensures that you do not make changes to the file that is used in recovery.

Database recovery

Recovery is the process of restoring your database file, transaction log, and dbspaces, and bringing the database file as up-to-date as possible with incremental transaction log files. It can be:

- **Automatic**: performed on database startup by checking if the database shut down cleanly at the end of the previous session otherwise by restoring all changes up to the most committed transaction
- **Manual** : recover a database to a previous version by applying one or more transaction logs to a copy of a database backup.

SQL Anywhere 17: Types of backups

Backup terminology	Explanation
Server and client side	Where the backup is executed . Faster on the server because data does not need to be trasmitted to another computer
Full and incremental	**Full**: copy of database files and the transaction log. **Incremental**: copy of the transaction log, they take much less time than full backups.
Image and archive	**Image**: copies and stores each file fo the database separately. **Archive:** format often used on tapes.
Online and Offline	**Online**: performed against a running database. **Offline**: just copies of the database files when the database is not running.
Live or continuous	A backup of the transaction log that runs continuosly while the database is running
Parallel	Server-side backup that makes use of the device-level parallelism to reduce the overall time required to complete the operation.
Snapshot	Compatible with Microsoft Shadow Copy Service (VSS)

Online, Offline and Live Backup

- **Online backup** : When an online backup process begins, the database server writes all cached data pages to the database file(s) on disk (checkpoint).
  - The database server continues recording activity in the transaction log file while the database is being backed up. The log file is backed up after the backup utility finishes backing up the database.
  - The log file contains all of the transactions recorded since the last database backup. For this reason, the log file from an online full backup must be applied to the database during recovery.
- **Offline backup:** The log file does not have to participate in recovery, but it can be used in recovery if a prior database backup is used.
- **Live backup:** provides a redundant copy of the transaction log for restarting your system on a secondary computer should the main database server computer become unusable.
  - A live backup does not stop. It continues running while the server runs. It runs until the primary server becomes unavailable. At that point, it shuts down, but the backed up log file is intact and can be used to bring a secondary system up quickly.

How to execute a backup

- SQL Command: (server-side)
  - Image backup
    BACKUP DATABASE
    DIRECTORY backup-directory
    [backup-option [backup-option ...]]
  - Archive backup
    BACKUP DATABASE TO
    archive-root[backup-option [backup-option ...]

    Example:
    BACKUP DATABASE DIRECTORY 'c:\\temp\\backup'
    TRANSACTION LOG RENAME;

- Backup utility (dbbackup):
  (client or server-side)
  - **dbbackup** [options] target-directory

    Example:
    dbbackup -c"Host=host;DBN=demo;UID=DBA;PWD=sql"
    SQLAnybackup

- Typical backup-options:

BACKUP DATABASE ...
TRANSACTION LOG ONLY
WAIT BEFORE START
WAIT AFTER END
WITH CHECKPOINT LOG {COPY | NO COPY | RECOVER | AUTO}
TRANSACTION LOG RENAME [MATCH]
TRANSACTION LOG TRUNCATE

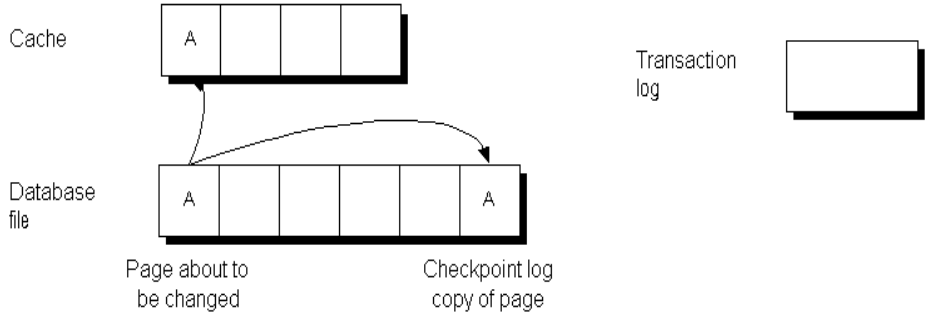
dbbackup …
-t [-x] #incremental
-wb
-wa
-k {copy| nocopy |recover | auto }
-r [-n]
-x #deletes the original log, starts a new one
-s #server-side
-l #live

Checkpoint log

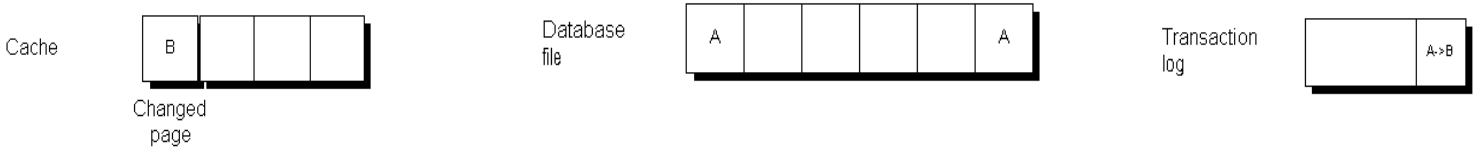
The checkpoint log is located at the end of the database file and is stored in the system dbspace.

Before any page is updated (made dirty), the database server:

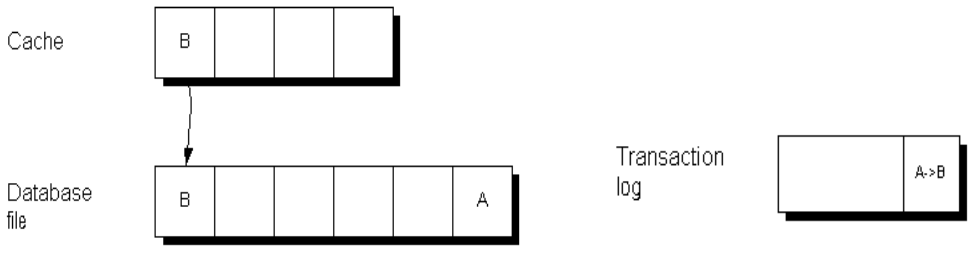
- reads the page into memory, where it is held in the database cache.
- makes a copy of the original page. These copied pages are the checkpoint log.



- Changes made to the page are applied to the copy in the cache. For performance reasons they are not written immediately to the database file on disk



- When the cache is full, the changed page may get written out to disk. The copy in the checkpoint log remains unchanged.



Checkpoint phase

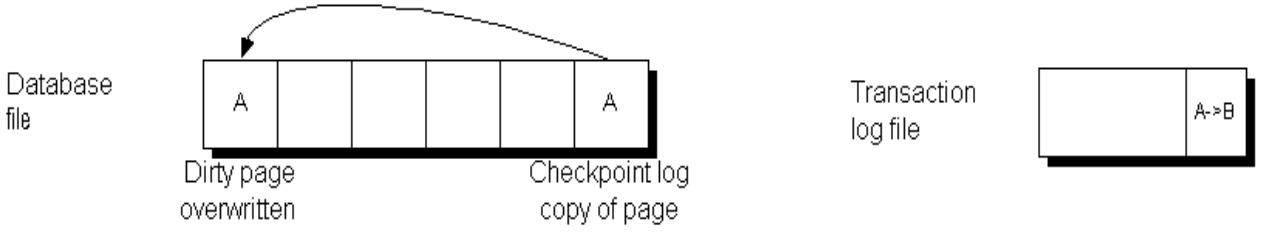
A checkpoint writes dirty pages to disk and represents a known consistent state of the database on disk.

- Following a checkpoint, the contents of the checkpoint log are deleted. As the checkpoint log increases in size, so does the database file.
- At a checkpoint, all the data in the database is held on disk in the database file. The information in the database file matches that in the transaction log. During recovery, the database is first recovered to the most recent checkpoint, and then changes since that checkpoint are applied.
- The checkpoint is activated by an explicit command (**checkpoint**) or implicitely by the following statements:
  - ALTER INDEX REBUILD , ALTER Table ,
    BACKUP DATABASE, COMMIT (when the database does not have a transaction log) ,
    CREATE DBSPACE , DROP DBSPACE ,
    DROP TABLE/ DROP MATERIALIZED VIEW statement (only if the table or view contains at least one row),
    LOAD TABLE , REFRESH MATERIALIZED VIEW , REORGANIZE TABLE, ...

The automatic recovery process

Each time that you start a database, the engine checks whether the last shutdown was clean(if the server performed a checkpoint) or the result of a system failure. If the database did not shut down cleanly, then it automatically takes the following steps to recover from a system failure:

- Recover to the most recent checkpoint .To restore all pages to their state at the most recent checkpoint, the checkpoint log pages are copied over the changes made since the checkpoint.



- Apply changes made since the checkpoint. Changes made between the checkpoint and the system failure, which are held in the transaction log, are applied
- Roll back uncommitted transactions.Any uncommitted transactions are rolled back, using the rollback logs

The transaction log (redo or forward log)

Is a file that stores all changes to the database. For example, inserts, updates, deletes, commits, rollbacks, and database schema changes are all logged.

When changes are forced to disk?

Like the database file, the transaction log is organized into pages: fixed size areas of memory. When a change is recorded in the transaction log, it is made to a page in memory. The change is forced to disk when the earlier of the following operations happens:

- The page is full.
- A COMMIT is executed.

- Completed transactions are guaranteed to be stored on disk, while performance is improved by avoiding a write to the disk on every operation.

How to execute a backup

- SQL Command: (server-side)
  - Image backup
    BACKUP DATABASE
    DIRECTORY backup-directory
    [backup-option [backup-option ...]]
  - Archive backup
    BACKUP DATABASE TO
    archive-root[backup-option [backup-option ...]

    Example:
    BACKUP DATABASE DIRECTORY 'c:\\temp\\backup'
    TRANSACTION LOG RENAME;

- Backup utility (dbbackup):
  (client or server-side)
  - **dbbackup** [options] target-directory

    Example:
    dbbackup -c"Host=host;DBN=demo;UID=DBA;PWD=sql"
    SQLAnybackup

- Typical backup-options:

BACKUP DATABASE ...
TRANSACTION LOG ONLY
WAIT BEFORE START
WAIT AFTER END
WITH CHECKPOINT LOG {COPY | NO COPY | RECOVER | AUTO}
TRANSACTION LOG RENAME [MATCH]
TRANSACTION LOG TRUNCATE

dbbackup …
-t [-x] #incremental
-wb
-wa
-k {copy| nocopy |recover | auto }
-r [-n]
-x #deletes the original log, starts a new one
-s #server-side
-l #live

WITH CHECKPOINT LOG

These options specifiy how the backup processes the database files before writing them to the destination directory. The default setting is AUTO for image backups and COPY for archive backups.

- **COPY** option : The backup reads the database files without applying any modified pages. The entire checkpoint log and the system dbspace are copied to the backup directory. The next time the database server is started, the database server automatically recovers the database to the state it was in as of the checkpoint at the time the backup started.
  Because pages do not have to be written to the temporary file, it can provide better backup performance,but it grows in the presence of database updates. Used if disk space in the destination directory is not an issue.
- **NO COPY** option : The checkpoint log is not copied as part of the backup. This option causes modified pages to be saved in the temporary file so that they can be applied to the backup as it progresses. The backup copies of the database files are the same size as the database when the backup operation commenced.
  This option results in smaller backed up database files, but the backup may proceed more slowly. Used when space on the destination drive is limited.
- **RECOVER** option : The database server copies the checkpoint log (as with the COPY option), but applies the checkpoint log to the database when the backup is complete. This restores the backed up database files to the same state (and size) that they were in at the start of the backup operation.
- **AUTO** option: The database server checks the amount of available disk space on the volume hosting the backup directory. If there is at least twice as much disk space available as the size of the database at the start of the backup, then this option behaves as if copy were specified. Otherwise, it behaves as NO COPY.

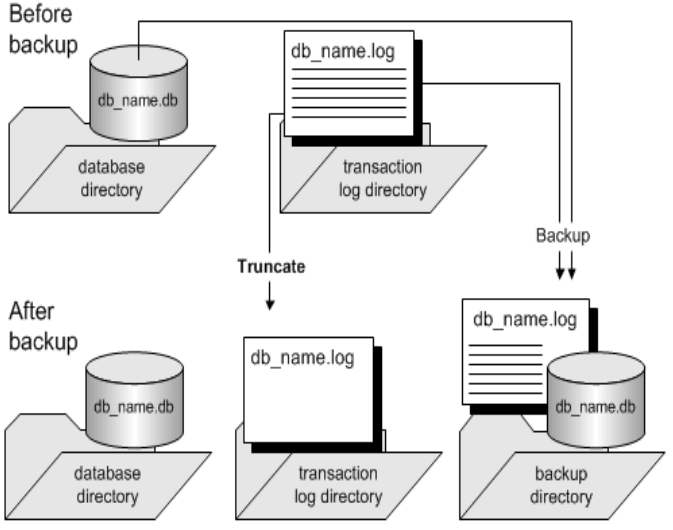
WAIT BEFORE START/AFTER END

- **BEFORE START** option
  This clause delays the backup until there are no active transactions. All other activity on the database is prevented and a checkpoint is performed.
  Using this clause with the WITH CHECKPOINT LOG NO COPY clause verifies that the backup copy of the database does not require recovery and allows you to start the backup copy of the database in read-only mode and validate it. When you validate the backup database, you do not need to make an additional copy of the database.
- **AFTER END** option
  This clause ensures that all transactions are completed before the transaction log is renamed or truncated. The database server waits for other connections to commit or rollback any open transactions before finishing the backup. Use this clause with caution as new, incoming transactions can cause the backup to wait indefinitely.

Transaction log rename

In addition to backing up the transaction log, a backup operation can rename the online transaction log to a file name of the form YYMMDDxx.log. This is called an offline transaction log.

- This file is no longer used by the database server, but is available for the Message Agent. A new online transaction log is started with the same name as the old online transaction log.
- The YYMMDDxx.log file names are used to distinguish between the files, not for ordering. For example, the renamed log file from the first backup on December 10, 2000, is named 001210AA.log. The first two digits indicate the year, the second two digits indicate the month, the third two digits indicate the day of the month, and the final two characters distinguish among different backups made on the same day.
- The Message Agent can use the offline copies to provide the old transactions as needed. If you set the delete_old_logs database option to On, then the Message Agent deletes the offline files when they are no longer needed, saving disk space.



Backup and Microsoft's VSS

- SQL Anywhere is compatible with the Microsoft Volume Shadow Copy Service (VSS)
- By default, all SQL Anywhere databases can use the VSS service for backups if the SQL Anywhere VSS writer (dbvss17.exe) is running.

SQL Anywhere and Microsoft's VSS

- VSS creates a point-in-time snapshot of the filesystem(s).
  - If VSS writer is NOT running, when the snapshot is taken, the state of the database files is equivalent to what it was at the time of the snapshot. Hence, the backup copy of the database has to go through the recovery process if the database server crashed before or during the snapshot.
  - If VSS is running when the snapshot is taken, it checkpoints the database and prevents modification to the database. Hence, the backup copy of the database is checkpointed and does not have to go through crash recovery. A checkpointed database does not require transaction log files for recovery.
  - VSS imposes time limits on how long it will allow any writer, including SQL Anywhere writer, to prepare its file before taking the snapshot. Therefore, it is still possible that the image of the database within the snapshot will not be checkpointed. This is why it is important to take backed up copies of the transaction log files.
- A checkpointed database contains the undo logs for uncommitted transactions. These are not rolled back during the snapshot. Hence, using the backup copy of the database it is possible to apply transaction logs generated after the snapshot/backup in order to recover those transactions as well.
- SQL Anywhere VSS writer does not do any post-backup processing. After the snapshot, the database can be modified.
- SQL Anywhere VSS writer lists all files (including dbspace and log files) associated with the databases that are running. Hence, VSS knows what files are needed to back up SQL Anywhere properly and a user of VSS (backup software) can take a correct snapshot.
- All restores are done using existing SQL Anywhere tools and VSS writer is not involved in the restore process.

Backup Strategies and data validation

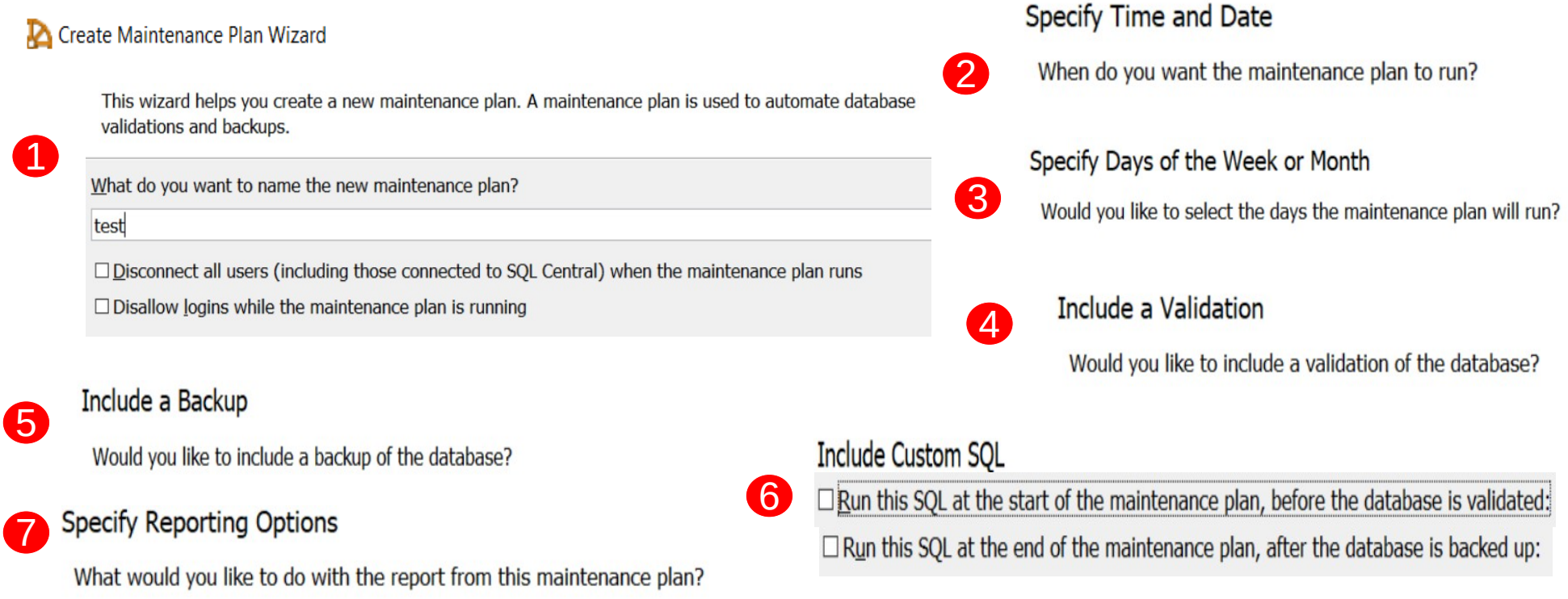
- **Backup is easy, recovery is hard...**

Design a backup & recovery strategy

- A good backup strategy incorporates a combination of full and incremental backups. Some of a key factors a dba should take in account are:
  - what files need to be backed up, where the database files are located, where the backup files are going to be stored ,
  - how many full backups you plan to keep around, whether the database server needs to remain available while the backup is running, whether to keep some of your full backups off site to protect against fire, flood, earthquake, theft, or vandalism, whether the database is involved in replication
  - how long your organization can function without access to the database (maximum recovery time in the event of media failure). External factors such as available hardware, the size of database files, recovery medium, disk space, and unexpected errors can affect your recovery time. When planning a backup strategy, allow additional recovery time for tasks such as entering recovery commands or retrieving and loading tapes, if needed.
- Maintenance plans automate validation and backup. You configure maintenance plans in SQL Central, and the types of maintenance tasks you can configure are:
  - Validation
  - Backup
  - user-defined operations (SQL statements) that precede a validation task or follow a backup task

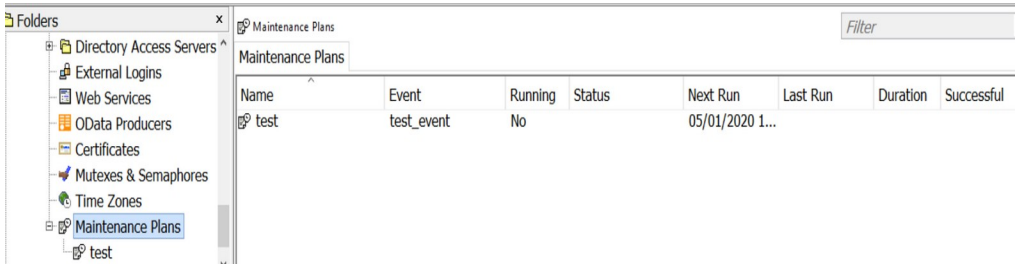
Maintenance plans

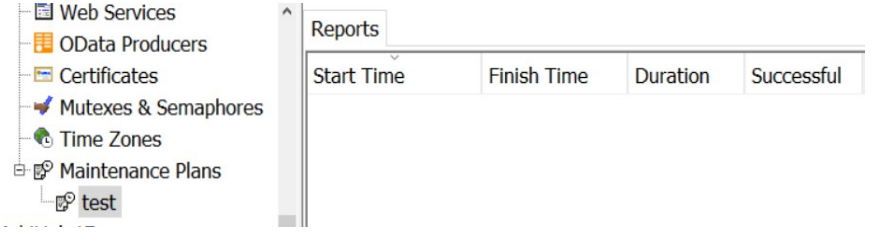
- **maintenance plan = event + report**
- Built with a SQL Central's wizard

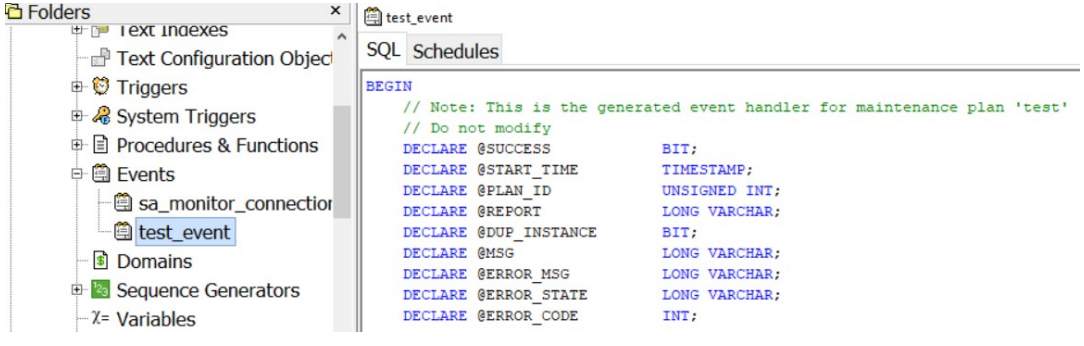


Maintenance plans

- The effect is:







Best practices

- Best practices recommend checking (validating) the data before making a backup, however an even better strategy is to perform the validation on the backup copy. These are the main reasons:

1) If you successfully perform the validation you can only assert that: the backup copy is valid and that the database is valid (or more properly it was valid at the time of the backup). Vice versa, if you validate the database you are running and then back up it, you cannot say for sure that the backup is valid
2) By running dbvalid in read-only mode on the backup copy you can avoid false validation errors triggered by any managed interventions (i.e. transactions in a production system)
3) You can run dbvalid on the backup copy on a different machine to avoid performance degradation
4) The backup and validation procedure can be automated and scheduled without interfering with normal operations on the production database.

Knowing that your backup is valid is more important than knowing that your running database is valid.

A possibly better way

In the following use case we try to implement a general strategy which can easily accomplish different use cases:

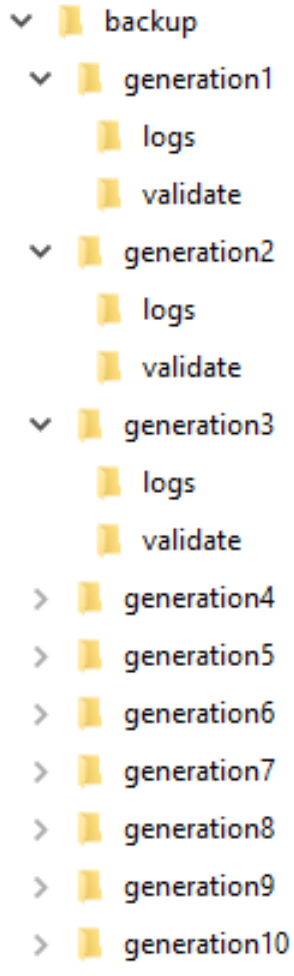
- It uses the command line interface so it can easily embedded on batch scripts
- It executes a configurable number of full backups intermixed with an inner loops of incremental backups
- Each full backup run involves a validation check to ensure that the backed up files are valid
- Each full and incremental backup involves the application of the backed up log files to ensure they will not cause any troubles in case of recovery
- Please, notice that: *I don't try to sell you somebody else's mule!* This strategy was proposed in a very old Sybase's blog (now Sap) you can stilll find at:

http://sqlanywhere.blogspot.com/2013/05/backup-is-easy-recovery-is-hard.html

Folder structure

In the following example we try to implement a general strategy which can easily accomplish different use cases:

- Each generationN folder contains all the activity related to a full database backup
- The logs folder contains all the backed up logs produced by the backup incremental command.
- The validate folder contains a copy of the full database backup. The validation and the backed up transaction logs are applied to this copy.
- The higher generation number holds the more recent backup



The use case

- In this use case the backup strategy plans to execute 10 full backups and for each single full backup 3 subsequential incremental backups:

```
REM Generation 10..
CALL full_backup.bat
CALL incremental_backup.bat
CALL incremental_backup.bat
CALL incremental_backup.bat

REM Generation 9...
CALL full_backup.bat
CALL incremental_backup.bat
CALL incremental_backup.bat
CALL incremental_backup.bat
...
REM Generation 1 ...
CALL full_backup.bat
CALL incremental_backup.bat
CALL incremental_backup.bat
CALL incremental_backup.bat
```

- We are running a test case scenario where a main batch contains sequential calls of technical batches (i.e , full_backup and incremental_backup.bat)

- In more realistic scenario the called batches are triggered by a scheduler.
  - For example consider a backup strategy use case where a full backup is scheduled every hour (starting from 10 am to 9 pm) and, after each full backup, 3 incremental backups are scheduled every 15 mins , i.e. x:15 , x:30 , x:45

full_backup.bat

- full_backup.bat implements the following sub-tasks:

Step 1 creation of temporary generation	RD /S /Q generation_temp ; D generation_temp
Step 2 Full backup	dbbackup.exe -c "ENG=ddd17;DBN=ddd17;UID=dba;PWD=sql" -o %LOGFILE% **-x** backup\generation_temp
Step 3 Copy of backed up db file(not the transaction log)	CD generation_temp; MD validate ; XCOPY ddd17.db validate /V /Q /K
Step 4 Transaction log's application	CD ..\.. ; REM Note: The -ad folder is relative to the folder containing the database. dbsrv17.exe -o %LOGFILE% backup\generation_temp\validate\ddd17.db^ **-ad** "%CD%\backup\generation_temp"
Step 5 Database validation	dbvalid.exe -c "UID=dba;PWD=sql;DBF=backup\generation_temp\validate\ddd17.db;ServerName=dbvalid;START=dbeng17 -c 20M -xd" ^ -o %LOGFILE% -q
Step 6 Generation folders rotation	RD /S /Q generation1 RENAME generation2 generation1 ... RENAME generation10 generation9 RENAME generation_temp generation10

incremental_backup.bat

- incremental_backup.bat works only on last generation10 and implements the following sub-tasks:

Step 1 Incremental backup

```
dbbackup.exe  -c "ENG=ddd17;DBN=ddd17;UID=dba;PWD=sql"^
  -o backup\generation10\dbbackup_log.txt  -n  -t  -x^
  backup\generation10\logs
```

Step 2 Transaction log's application

```
 REM Note: The -ad folder is relative to the folder containing the database.
dbsrv17.exe  -o backup\generation10\dbbackup_log.txt^
  backup\generation10\validate\ddd17.db^
  -ad "%CD%\backup\generation10\logs"
```

Simulating transactional workload

- In order to approximate an heavy used db server, the following sql statements are add to a sample dattabase:

Sample table

```
CREATE TABLE rapid (
   pkey   INTEGER NOT NULL DEFAULT AUTOINCREMENT PRIMARY KEY );
```

Event which execute one loop where one row is inserted every 1/10 of a second

```
CREATE EVENT rapid_insert
HANDLER BEGIN
   WHILE 1 = 1 LOOP
      INSERT rapid VALUES ( DEFAULT );
      COMMIT;
      WAITFOR DELAY '00:00:00.1';
   END LOOP;
END;

TRIGGER EVENT rapid_insert;
```

In any moment you can check if the sample table has kept pace with transactions during the test backup generations loop

```
"%SQLANY17%\bin64\dbisql.com"^
  -c "ENG=ddd17;DBN=ddd17;UID=dba;PWD=sql"^
  SELECT COUNT(*), MAX ( pkey ) FROM rapid

REM PAUSE Note: COUNT should equal MAX
```

Database recovery

- Suppose you have just completed the entire loop with 10 generation runs and the production database crashes by corrupting the database file

```
COPY backup\generation1\ddd17.db

REM Note: -o writes to the dbbackup diagnostic file in this example.
REM Note: -ad is relative to the folder containing the database.

dbsrv17.exe -o dbrecover_log.txt ddd17.db -ad backup\generation1
dbsrv17.exe -o dbrecover_log.txt ddd17.db  -ad backup\generation1\logs

dbsrv17.exe -o dbrecover_log.txt ddd17.db  -ad backup\generation2
dbsrv17.exe -o dbrecover_log.txt ddd17.db  -ad backup\generation2\logs

……

dbsrv17.exe -o dbrecover_log.txt ddd17.db  -ad backup\generation9
dbsrv17.exe -o dbrecover_log.txt ddd17.db  -ad backup\generation9\logs

dbsrv17.exe -o dbrecover_log.txt ddd17.db  -ad backup\generation10
dbsrv17.exe -o dbrecover_log.txt ddd17.db  -ad backup\generation10\logs

REM Use dbsrv17 -a to apply the most recent log.
dbsrv17.exe -o dbrecover_log.txt ddd17.db -a ddd17.log
```

- Note: this scenario is the worse one. You recovered all the logs from the oldest backup.In the best case you can just recover from the more recent backup (generation_10) or from an intermediate generationX folder

```
COPY backup\generation7\ddd17.db
dbsrv17.exe  ddd17.db  -ad backup\generation7
dbsrv17.exe  ddd17.db  -ad backup\generation7\logs
 ……
dbsrv17.exe ddd17.db  -ad backup\generation10
dbsrv17.exe -t ddd17.db  -ad backup\generation10\
logs
dbsrv17.exe -o dbrecover_log.txt ddd17.db -a
ddd17.log
```

SQL Anywhere Backup, Validation & Recovery

- Further readings
  - *https://wiki.scn.sap.com/wiki/display/SQLANY/Backup%2C+Validation%2C+and+Recovery*
- This presentation will be available to all the participants

- For further information you can contact me by email:

luca.casavola@softpi.com