American Journal of Database Theory and Application 2021, 6(1): 1-15
DOI: 10.5923/j.database.20210601.01

An Effective Database Change Management System

Shirish Patil

Database Administrator / Lead Enterprise Data Architect, Sitek Inc, USA

Abstract Change Management Systems or a version control mechanism is the pillar stone of success for any software development effort. Rapid changes and concurrent streams of development due to business complexity, legal, or any other reason needs the best version control mechanism to manage these changes. Business changes more often than not results in modifications for the underlying database system as that is where the data is stored. Very few organizations realize that how difficult it is to undo a database change in a relational database management system (RDBMS) [1]. If by mistake an incorrect version of the application code is deployed, the correct version can be re-deployed and for the most part, everything will be fine. Whereas incorrect database change can be destructive, incur partial data loss to a complete data loss, or even result in database restore. That is a huge cost to pay in absence of effective change management system. In today's software organizations, time-to-market is paramount [2]. To achieve this, organizations develop software in a highly aggressive schedule with multiple streams (or releases) being worked upon concurrently. This provides a huge challenge to ensure that only the exact change(s) are deployed in various Development, Test, Performance, Pre-Production, and Production environment. This is where a Database Change Management System can help and this document explains exactly how that can be done.

Keywords RDBMS, Database, SDLC, Change Management, Data Cloning, CI/CD

1. Introduction

The software development process involves developing programs using a programming language to implement business requirements so that the services can be delivered to the users. These delivered pieces of software or services involves storing and processing information and data. This data is stored within the data structures or databases, which can be a relational database or a NoSQL database. Managing the changes to the data structures (database tables and columns) on how the data will be stored or retrieved is extremely challenging with concurrent software releases and a large number of changes. For this very reason, managing the database changes in an efficient manner is extremely critical for the success of any software development organization. This work aims at addressing these challenges and how to effectively manage with a flexible, license-free, feature-rich solution.

2. Challenges in Software Development

Before diving into the Database Change Management Solution, let's briefly understand the setup of a software organization and how the actual development and deployment takes place. Every business requirement is developed into a software code which then eventually gets released or deployed into a production version of the application used by end-users.

Most software development organization will have these environments [3]: Local, Development, System Integration Test (SIT), User Acceptance Test (UAT), Performance Test/Load and Stress Test (LaST), Pre-Production, and Production. Each one of these environments have a unique purpose and most often being used by different set of users/teams. Local and Development environments are mainly used by teams writing the software programs, converting the business requirements into technical solutions. SIT and UAT are the environments used by testing teams to ensure that the developed solutions meet the business requirements as documented. Performance test or LaST environment is used to ensure that the solution is scalable under higher load, pre-production is used to do a dry-run of production deployment of the changes. The production environment is where the final set of tested and approved changes are deployed for end-users to use. What change or number of changes a particular environment is ready to accept depends on the team capacity, urgency of business requirement as well as the readiness of the change itself. It is very common for each of the environments, except production to be at a different set of changes. Managing these different sets of changes and deploying only what is required and approved to be deployed is enormously challenging.

* Corresponding author:
patil.shirish@outlook.com (Shirish Patil)
Received: Jan. 21, 2021; Accepted: Feb. 3, 2021; Published: Feb. 6, 2021
Published online at http://journal.sapub.org/database

Multiple teams work on these environments simultaneously with a primary goal of delivering quality software for end-users. They develop changes which have to be implemented across these environments. It becomes extremely critical for the organization that the correct list of changes gets implemented to the correct environment. Depending on the software development methodology used; Waterfall, Agile or Hybrid, there are unique challenges faced by technology teams.

Software development is an evolving process [4]. Depending on the business and scale of the organization, there may be one team working on several sets of environments or many teams working on several set of environments. Code development and test process is iterative and should be completed well ahead than the actual production deployment day. This gives the technology group a window to deploy and test the production ready code in a pre-production environment for identifying/resolving any issues upfront, before the code is deployed for end-users to use. This process is called a dry-run. In the absence of this process, a failure in production environment can render the product or services unusable for the end-users leading to huge financial losses or loss of reputation [5].

Let's understand this with an example of an organization deploying 6 major releases to production in a year, each one in approximately 2 months. Figure 1(a).

Every 2 months, on the marked date above, developed and tested features for that release will be deployed to production. It is not necessary that this development and testing of the code will only begin after the end of the previous release. More often than not, the development and testing efforts overlap with each other. Figure 1(b) explains it.

This is where the complexity is. These simultaneous efforts, either by the same set of resources, or different, need a highly specialized skillset in ensuring that the environment where code and database changes are to be deployed is available for the teams to work on. Any downtime is a huge cost to the organization as the average cost of downtime is $9,000 per minute [6]. Version control systems play an important role to ensure only the intended changes are implemented. Agile software development exponentially increases the complexity as many sprint teams can work on the same set of database objects when developing for the associated user stories. These user stories may be directly delivering the features and functionality for the services of the organization which in-turn may impact the revenue or goodwill of the company.

Database change management can make-or-break in such scenarios and impact the service delivery capabilities. This solution addresses and solves the challenges discussed above and enables the organization efficiently manage the most critical aspect of their system development lifecycle. This solution is intended to provide flexibility, enhance capabilities, save cost and deliver database changes accurately and efficiently.

Figure 1(a). Release Dates for Production

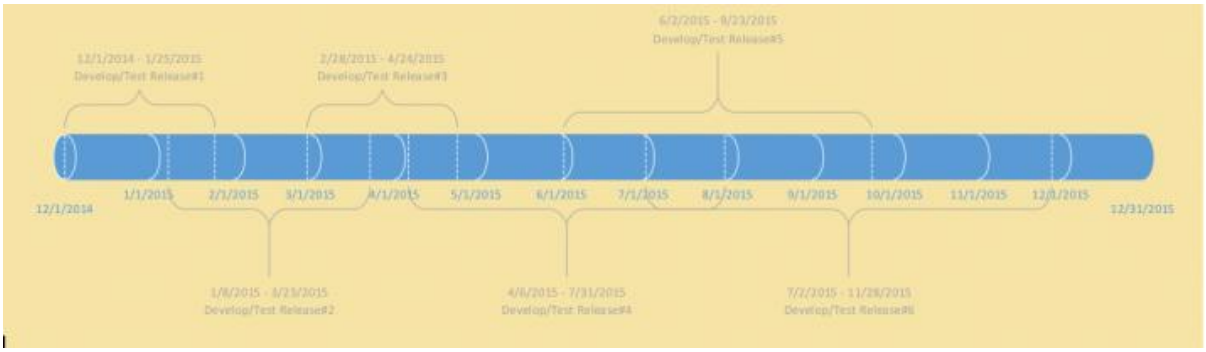

Figure 1(b). Development and Test Timelines

3. This Solution

Database is the backbone of any software [7]. Data is stored and processed within the database. Better design and architecture of a database results in application scalability and performance to meet the Service Level Agreements – SLAs. For small databases with few objects, it may be ok (although error-prone) to have minimum version control, but when the number of objects grows drastically resulting from exponential organization growth or rapid requirement changes, it becomes difficult to manage changes on the database. These events can result in several new database objects being added to a release or changes into existing objects or a mix of both. Any of these changes may significantly impact the performance of an application and an incorrect change implementation can disrupt organization's ability to deliver its services to customers. The disruption can cause loss of revenue and may impact organization's reputation as well. *To mitigate such risks, a sound database change management system is needed to ensure that only the intended changes are deployed. That is exactly what this solution does*.

With its rich features, flexibility to customize, and cost-effectiveness, this solution immediately adds value to the software toolset of any organization. The following sections will elaborate on the features, framework, architecture, algorithm of the solution, and how-to configure, implement and maintain this solution. We will also understand the benefits of the program and will conclude with an understanding of which type of organizations are best suited to use this solution.

3.1. Features

Features are a major differentiator for any software solution. When procuring a software product from external software vendors, companies often compare features of one over another to decide which one to buy and which is best suitable to fulfil their requirements [8]. The solution having rich features and ease of use are in high demand. Such solutions enhance the organization's capabilities to deliver services to their end-users much quicker and efficiently.

This Database Change Management System is a very simple yet highly effective solution and has extremely desirable features: Cost-Effective, Change Tracking, Requirement Traceability, Change Inventory, Customization and Replication, Master List and Auditing, Environment and Application Specific, Automation, Compatibility with External Tools, Database refresh compatible, Continuous Integration Continuous Deployment (CI/CD) and Training Dataset Plug-in.

Each of the features listed above are highly desirable and needed by the software development teams in accurately developing and deploying the database changes across multiple environments and releases. The majority of the database teams manage the changes manually and deploy the database scripts manually. The scripts are stored in the folders and then picked up manually to execute during the deployment. Manual execution is always error-prone when the teams are dealing with a large number of changes or even with a small number of changes but a large number of environments.

Following sections discuss in detail about the features of this solution.

3.1.1. Cost-Effective

Whenever an organization evaluates commercial software they need to work within the approved budget as the licensing, maintenance, and support cost can take away a significant chunk of their budget. This solution resolves the issue by being a no-cost solution for the organization. The two components of the solution, scripting language and the database, are flexible as such that the companies can use any scripting language they currently use or an open-source scripting language. The database can be any relational database currently being used in the organization, which can act as the repository for the master data and configuration data for the solution. Such a robust solution with no cost associated with it is a distinct advantage for the companies as the saved funds can be repurposed for other initiatives.

3.1.2. Customization and Replication

Sometimes an application may need custom changes which does not apply to other application's database change management solution. That is where this solution stands out. Each application has its own version of application-specific executable database change management solution. Any custom changes can be made to the specific version without impacting other versions. This outstanding capability provides superior flexibility to the teams as they have the freedom to enhance and modify the solution as required. As the solution is developed in the native scripting language and a relational database, customizing the solution is very easy.

When an organization starts developing a completely new application or a new module, the technology teams can easily replicate the code and configuration of this database change management solution to create a new version for the new application. This replication is rapid, easy, and less complex. Organizations can replicate as many times as needed and for as many applications as needed. There is no limitation on the number of versions created. Again, a distinguished feature to help organizations with the cost impact of customization. With a licensed product, any such customization would incur cost, per new application.

3.1.3. Change Tracking with Requirement Traceability

Each change is assigned an intelligent number which in itself explains the release and how many changes are in the release up to this number. For example, if it is a 16th change in the 51st release of the application, the build number for this change will be 5100016. The last 3 digits in this numbering scheme belong to the number of changes in a

particular release, with the first 2 being the release number. 3rd and 4th digits are for intermediate releases such as 51.1 or 51.1.1.

When a particular build is created it is always tagged with the business requirement number for traceability. With hundreds of changes being developed for multiple releases having this technical change to business requirement mapping always helps in understanding why a particular change was implemented.

3.1.4. Change Inventory, Master List, and Auditing

The metadata belonging to every change for each release is stored in the master repository. Each application gets its own set of tables to store metadata and hence stays completely independent of each other. The master list of all the changes is used to compare with the changes in the specific environment and the difference defines what are the pending changes to be applied. Implementation details of each environment is stored and hence provides critical auditing details as to when the changes were implemented, how long the changes took, etc. The auditing information is captured for each change with the log of execution on a specific environment, which provides critical root-cause analysis functionality when the teams are trying to resolve a particular issue.

3.1.5. Environment and Application Specific

Each application gets its own set of scripts. This helps in keeping application-specific changes to the tool contained to itself and independent. There are checks and balances inbuilt into the solution so that the scripts designed for application A will not execute on application B and vice-versa. The scripts can be easily enhanced to implement an application-specific change to its own set of tools. The following image (figure 2) depicts how changes are propagated across a single application environment for a particular release.

Development teams need control to implement specific changes to specific environments only when they are required. This tool can apply selective changes to selective environments. An outstanding feature that is not often found in such no-cost solutions.

3.1.6. Automation

The scripts can be manually executed to apply the required number of changes. Only the highest build number needs to be provided as a parameter to the tool and internal automation will check the number of builds created for the application, compare them with the builds already implemented in the target environment and then generate an executable script to apply the pending builds, up to the provided build number. This automation helps in applying all the builds in single execution vs applying each one individually.

3.1.7. Compatibility with External Tools

Since this solution is developed using a scripting language, it is very easy to integrate it with other external tools. A simple call to the executable script along with required parameters, will be able to run this tool from another software.

3.1.8. Database Refresh Compatible

As part of the testing initiatives, most often, environments like User Acceptance, Integration Test and Performance Test needs data from the Production Environment. A very common method used to ensure the test environments are production-like environments are testing produces good results, the test environments are refreshed using a copy of production environment. Generally, the production environment does not have changes that are currently being developed for future releases and that need testing. When such refreshes happen, it also overwrites the metadata within the environment from the one in the production. This change management solution has inbuilt capabilities to manage and resolve such issues by re-configuring the metadata within the environment to appropriate levels.

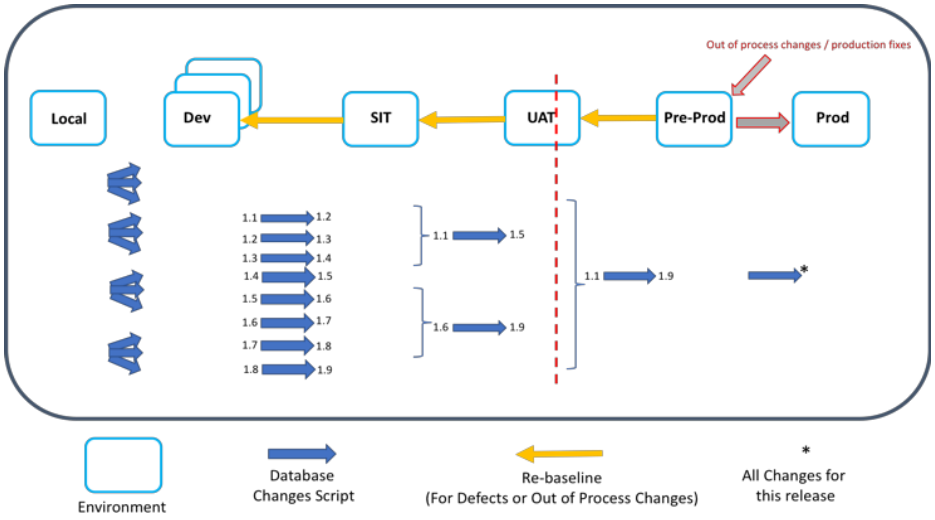

Figure 2. Database Change Implementation for Single Application Across Environments

3.1.9. CI/CD

Continuous Integration Continuous Deployment (CI/CD) is becoming increasingly popular [9] and this database change management solution is completely compatible with CI/CD processes and tools. The executable can be directly called by the CI/CD tools during the deployment. The tool has built-in capabilities to generate logs, identify success or failures and send notifications, which is an integral feature for continuous integration continuous deployment process.

3.1.10. Data Cloning

Training employees on new features of the software application/services is integral to the success of any organization [10]. Specifically, within government agencies, when a new version of the software is released the client or citizen-facing employees need to be trained on the new features so that they can efficiently help the citizens. To get trained in a real-life like scenarios, the application needs to have real-life like datasets. This database change management solution has a plug-in that can be integrated within the application to generate new datasets by completely obfuscating PII data such as SSN, DOB's. The same type of dataset can be generated for all the trainees of the same class as they are all trained at the same time, which allows the instructor and trainees to have a constructive training session and clear doubts about the new features upfront. This process works based on a golden data set called a golden case, which is used to produce multiple new copies of the same dataset with new identifiers.

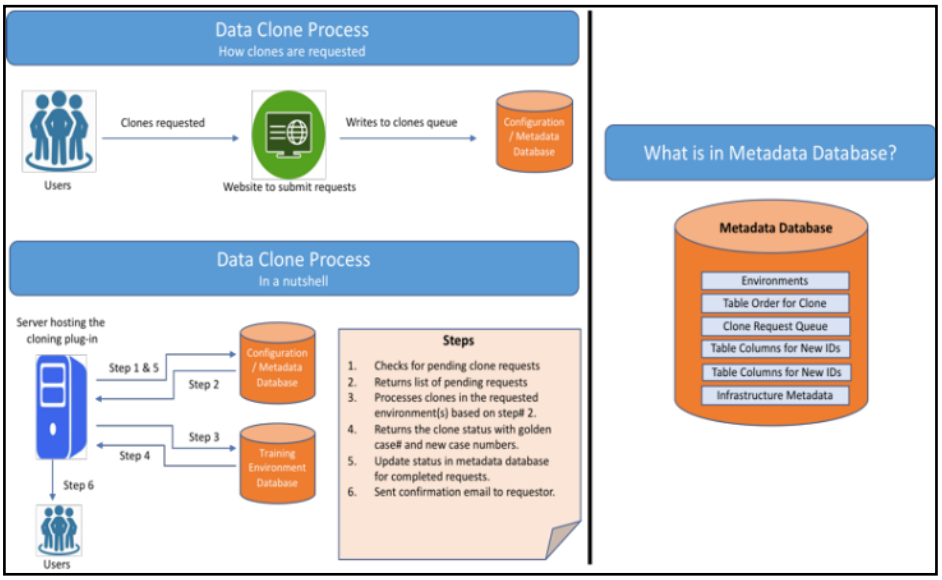

Figure 3(a). Conceptual Representation

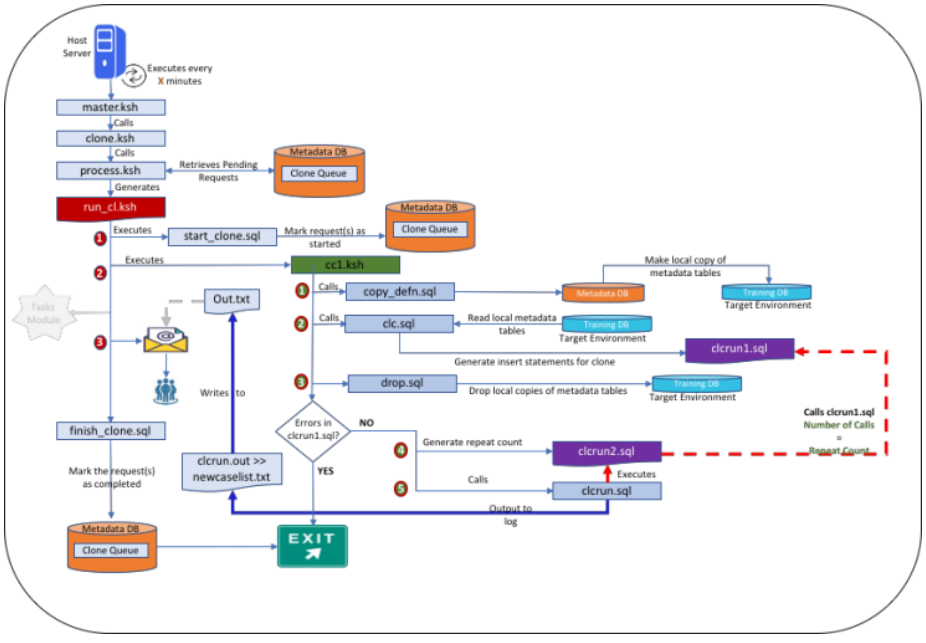

Figure 3(b). Overall Flow of the Process

Figure 3(a) illustrates a conceptual representation of how this plug-in works. This includes how users can request to clone data after specifying the environment and number of clones, their email address. The webpage also provides capability to schedule the clones in the future. This is especially helpful when there are a large number of training sessions scheduled and datasets need to be prepared. The plug-in processes each of the requests sequentially and scheduling capabilities help to mitigate the limitation due to sequential processing.

3.2. Framework

This solution needs one master database to host the related metadata. This framework can be accessed to identify the configuration of any environment, or even all environments for a specific release. Some applications are a mix of individual separate applications that communicate with each other in terms of finally delivering the functionality to end-user.

This individual metadata can be categorized as following:

a. Database
b. Server
c. Schema's or Database Users where objects will be hosted
d. Peripheral users to maintain security: Read-Write or Read-Only users
e. Application
f. Environment: Dev, Test, Performance, Pre-Prod, Production, Training etc
g. Application-Database Change Master list for each application

3.3. Algorithm and Code

An algorithm is a finite sequence of well-defined, computer-implementable instructions, typically to solve a class of problems or to perform a computation [11]. Algorithms are always unambiguous and are used as specifications for performing calculations, data processing, automated reasoning, and other tasks. The algorithm for this database change management solution explains in detail the logical steps of the solution. This Database Change Management Solution is a two-step solution: Step 1 is to package a particular database change or changes into a single build and Step 2 is the deployment process where one or more of these packaged database builds are deployed. Let's understand each of these steps in detail.

Setting up the database build is the first step in the process. During a software development cycle, business requirements are discussed and converted into technical implementations which include software code and database code. Finalized database design for a particular business requirement results from several discussions between data architects, developers, business analysts, and functional teams. This finalized database design is a must for the code to work without errors and for delivering the required functionality. Above (figure 5(a)) is the pseudocode for how the finalized database change is packaged into a build that is error-free and ready to be deployed. These simple steps allow the users to quickly utilize the pre-built code and only making minimum changes that are specific to this new database build.

While creating the new build most of the logic and identifiers for audit and logging purpose is already updated by the tool. The only manual changes required are the executable SQL statements related to the new database built. Figure 5(b) represents the code that is used to create a new database build for the application called APP1, where the executable located, and what are the contents of a sample build. Figure 5(c) has the code which gets executed (setupbuild.ksh) to create a build.

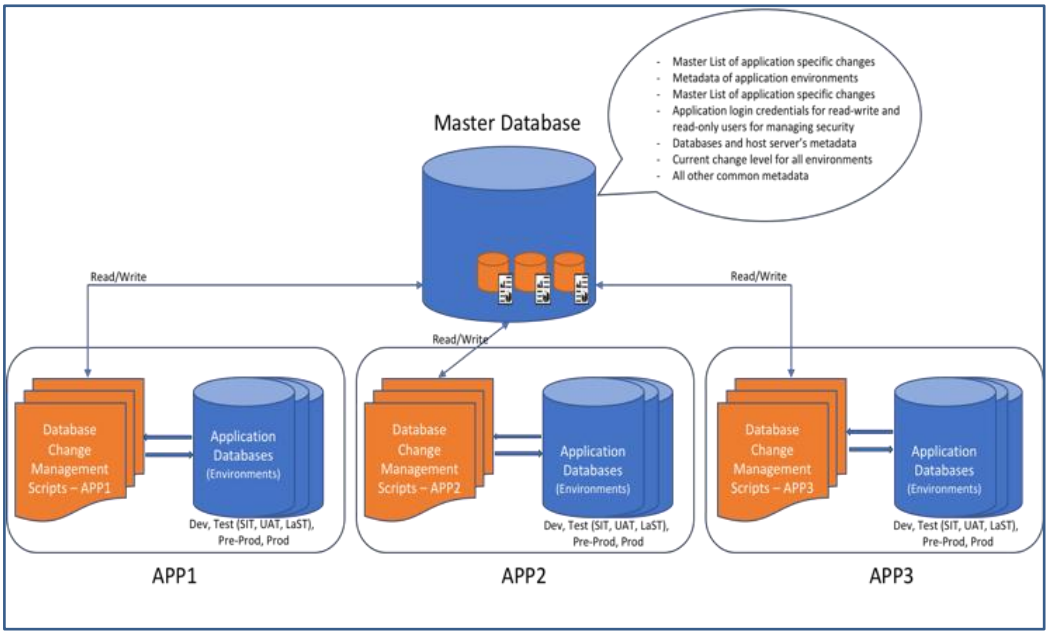

Figure 4. Representative Framework for 3 Applications

Creating a build for a single change or a set of changes

Script: setupbuild.ksh
Location: on the server, go to the application specific directory under *dbchanges*
Parameters required: SourceBuild#, TargetBuild#, BusinessRequirement#

1. Copy all files from SourceBuild#. There are two set of files: KSH which is executable and SQL which contains the sql statements for the database changes.
2. Create a new directory and name the directory as TargetBuild#
3. Paste all file copied from SourceBuild# into TargetBuild# directory.
   a. Changes build# within all the files
   b. Changes create date within all the files
   c. Changes BusinessRequirement# to the new one within all the files
   d. Updates Dev environments change date to current date.
4. Update executable file with build#, New Tag and Sql file name.
5. Rename the sql file with the name in the executable file.
6. Remove old sql statements from the sql file and replace with the sql statements which belongs to this build#.
7. Check all file for errors and fix errors, if found.
8. Once all the errors are resolved, build is ready to be applied.

Figure 5(a). Algorithm/Pseudocode for Creating Database Build

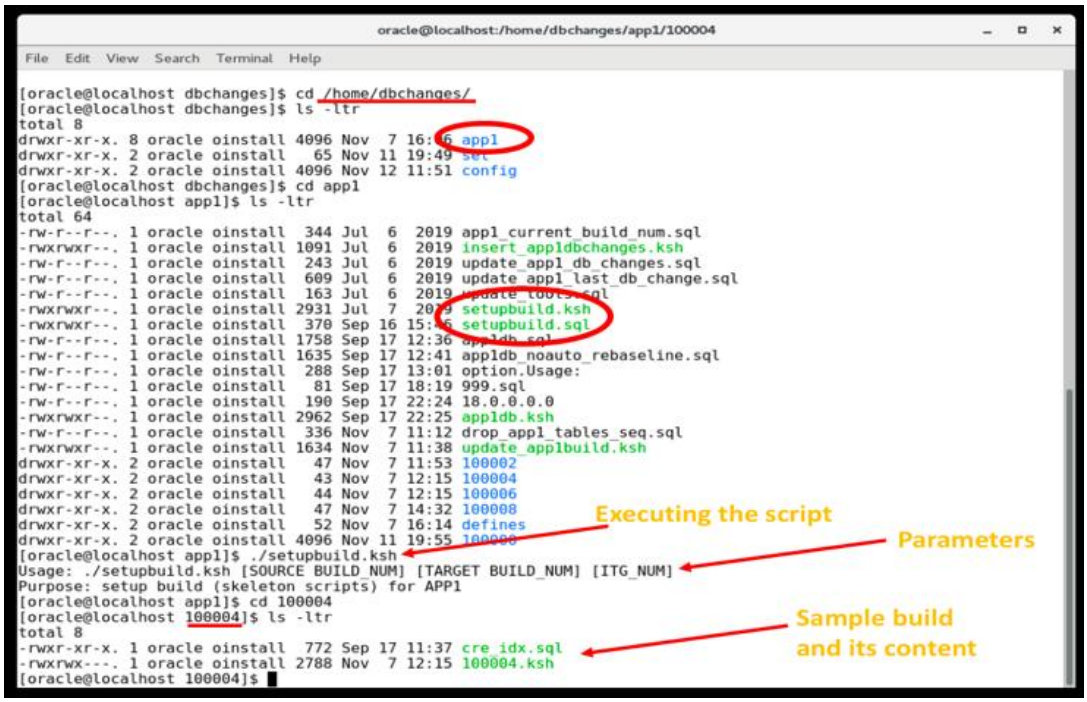

Figure 5(b). Database Build Creation Scripts, Directory & Sample Build

Once the database build is created, the files are checked for any errors and the errors are resolved, the builds are ready to be applied to any environment for that application, APP1 in this case. If the organization is developing multiple independent applications then each of these applications will have a separate directory under the "*/home/dbchanges*" directory. There is no limitation on how many applications can be supported using this solution. This method allows separation of database changes for each application and allows the team for easy maintenance and search capabilities, which is a must if there are 100's of changes being created/implemented per application per release.

The second part of this solution allows multiple database changes to be implemented in a single execution. *This step is further subdivided into two parts: the first one is the implementation of the required number of changes (database builds) and the second part is to set/reset the access privileges for authorized database and application users.* As discussed in section 3.1.5, each environment has varying needs of how many changes can be accepted by the respective team, even when they are of the same application and same release. The development team may continue

developing new changes in the development environment(s) whereas Integration and Testing teams may not be ready for all the changes which are developed. Hence there is a need to only deploy the changes in an environment when that specific team is ready for the changes. *The number of changes can be either selective changes ranging from very few to all of the changes developed.*

This database change management system allows the user to specify the target build number and automatically prepare the executable for implementing all the changes from the current build number of the environment to the target build number. *It is important to remember that the changes are applied sequentially. If the user intends to disable or not apply a particular change to a particular environment, then it needs to be turned off from the generated executable for that environment.*

The tool has internal steps to verify few items before it starts executing. These checks and balances are very critical for the integrity of the environment. Any implementation of a non-required change in the environment can lead to downtime in the environment and even to a complete restore of the database. The framework starts first with ensuring that the application on which the database change management system is being executed is of the same type for which the system was created. This allows to prevent inadvertent implementation of a change of one application to another. Once this check is passed, the framework checks for the current build number within the environment, and the build number for that environment in the master configuration matches.

Figure 5(c). Representative Code for Creating a Database Build

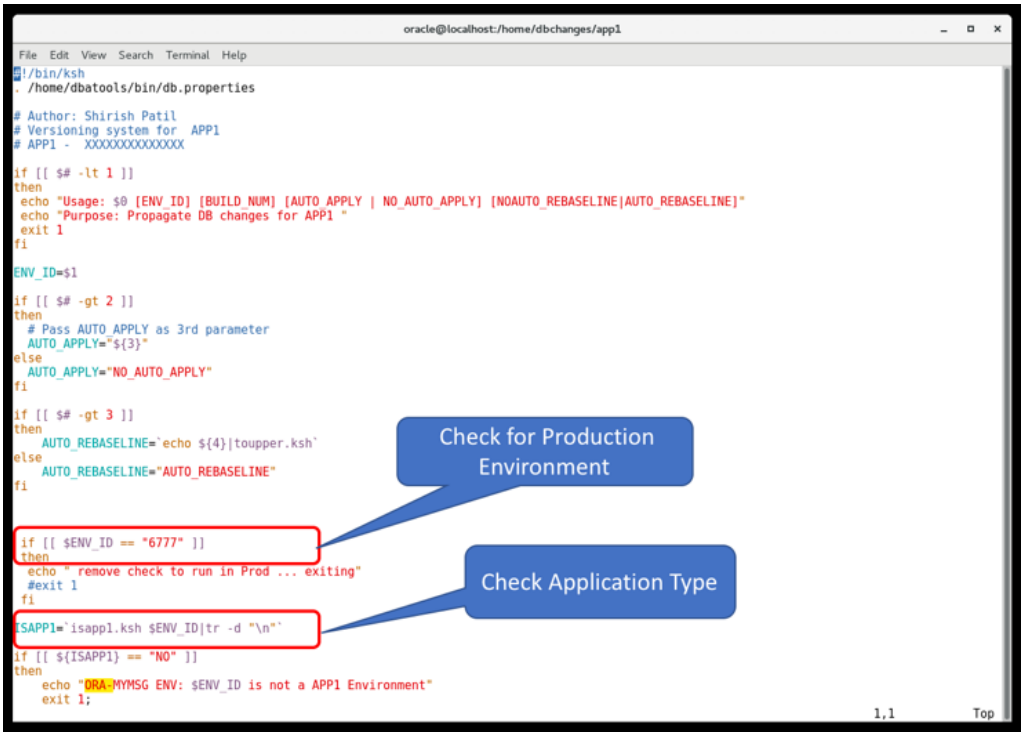

Figure 5(d). Check for Production Environment and Application Type

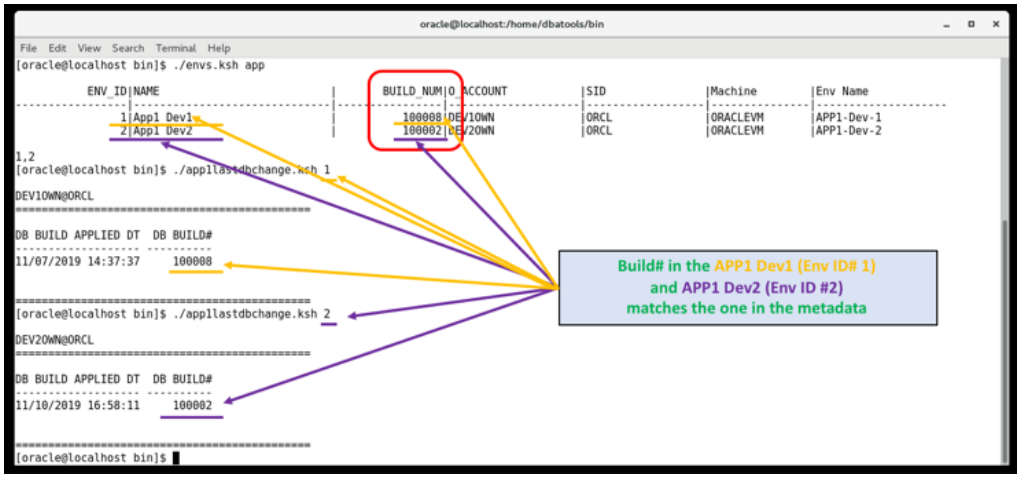

Figure 5(e). Build# Matches Between What is in Environment vs. Master

Only when these two preliminary checks are passed, the framework proceeds with generating the executable script for the number of changes requested. There are additional checks to prevent the execution in a production environment. Each one of these prevention and protection mechanism are configurable as we can see in the figures 5(d) and 5(e). In the event of a build# mismatch between the Metadata DB and the application environment or if the application type is not the same as the application for which this database change management system is built, the script will terminate. The application type mismatch issue can be resolved simply by using the correct database change management system. Build number mismatch issue can occur if the application environment was refreshed from another and the build numbers were not reset. Database refresh is a very common occurrence where testing environments are often refreshed from a production environment to test with real data. In such a scenario, the build number of the production environment will be in the test environment and hence the build number for the test environment in the Metadata DB will need to be reset to match with what is in the environment. After this correction the execution of the scripts can proceed.

Implementing the requested number of changes works on the basic principle of Change Management. In any change management system, there is a Master List which contains the list of all the changes and then there are environments or components where these changes are to be applied. Once the user provides the required target change number for the environment, the Change Management system compares the target change number with the master list and current change

number of the environment and provides the required list of changes to be applied. This comparison is the core logic of any change management system and plays a vital role in this database change management system as well. The master list of database changes(builds) is stored in the Metadata DB. Once created this list is untouched for updates, maintaining the integrity of this foundational component of the database change management system. Only new changes are added to this list as and when they are created.

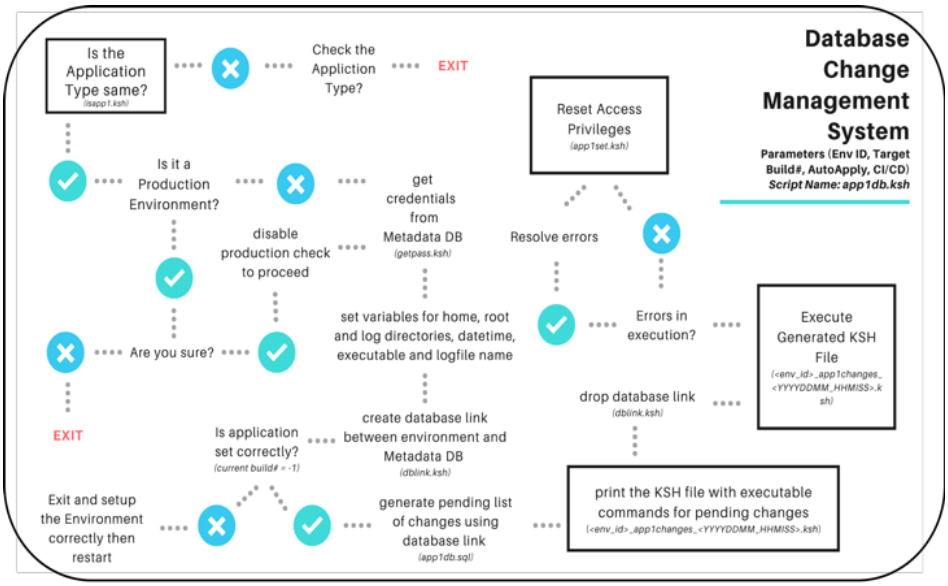

Figure 5(f). Flowchart of Database Change Management System Process

```
TOOLSLOG="/home/dbalogs/app1/changes"
TS=`date +%Y%m%d_%H%M%S`

KSHFILE="${TOOLSLOG}/${ENV_ID}_app1changes_${TS}.ksh"
LOGFILE="${TOOLSLOG}/${ENV_ID}_app1changes_${TS}.log"
TEMPLOGFILE="${LOGFILE}.tmp"
ALLBUILDSLOG="${TOOLSLOG}/${ENV_ID}_app1changes_all_${BUILD_NUM}.log"

touch $LOGFILE
touch $TEMPLOGFILE

dblink.ksh del dbapp1 999 $ENV_ID
DBLINK=`dblink.ksh cre dbapp1 999 $ENV_ID`
DBLINK="dbapp1_$ENV_ID"
CURRENT_BUILD=`echo $OWNP |$SP ${OWNN}@${OWND} @${TOOLSROOT}/app1_current_build_num.sql|tr -d "\n"`
echo
echo
echo ORA-MYMSG CURRENT BUILD NUMBER is $CURRENT_BUILD

if [[ $CURRENT_BUILD == "-1" ]]
then
  echo "APP1 ENVIRONMENT NOT SETUP, COULD NOT QUERY AA_APP1_LAST_DB_CHANGE table"
  echo "Start with running ${TOOLSHOME}/100000/100000.ksh"
  exit 1;
fi

echo ORA-MYMSG APPLY APP1 BUILD IN $ENV_ID >> $LOGFILE
echo " " >> $LOGFILE

if [[ $AUTO_REBASELINE == "AUTO_REBASELINE" ]]
then
echo $TOOLSPASS|$SP $TOOLSOWN @$TOOLSROOT/appldb.sql $ENV_ID $BUILD_NUM $CURRENT_BUILD $DBLINK $KSHFILE  | tee $TEMPLOGFILE
else
echo $TOOLSPASS|$SP $TOOLSOWN @$TOOLSROOT/appldb_noauto_rebaseline.sql $ENV_ID $BUILD_NUM $CURRENT_BUILD $DBLINK $KSHFILE  | t
fi
echo " " >> $LOGFILE
cat $TEMPLOGFILE >> $LOGFILE
echo
echo
echo
echo
echo
BUILDS_TO_PROCESS=`grep 'NO OF BUILDS TO PROCESS:' $KSHFILE|awk -F: '{print $2}'`

if [[ $BUILDS_TO_PROCESS -gt 0 ]]
then
  echo
  echo
  echo
  echo TO APPLY ALL CHANGES
  echo    "EXECUTE COMMAND: nohup /bin/ksh $KSHFILE > $ALLBUILDSLOG 2>&1 & "
  echo
  echo   "THEN TAIL THE LOG (for all builds): tail -f ${ALLBUILDSLOG}"
  echo
fi


if [[ $AUTO_APPLY == "AUTO_APPLY" ]]
then
  /bin/ksh $KSHFILE
fi

dblink.ksh del dbapp1 999 $ENV_ID

"appldb.ksh" 133L, 2962C
```

Figure 5(g). Database Change Management System Driver Script

The script also has some additional in-built sub-modules to process the request seamlessly without any manual intervention from the user. Some examples of such sub-modules are: password extraction for the application schema, setting up the log file location, setting up the logfile name for audit purposes, setting up database link from the environment to the metadata database, cleaning up database link, auto-apply of the final generated executable file. Here is the flow chart (figure 5(f)) of how each component of this system interacts with each other and how the system works. Figure 5(g) has the representative code of the database change management system driver script.

As explained in the above flowchart (figure 5(f)), if the process is completed and all the errors (if any) are resolved, then the last step will be to ensure that the changes implemented are accessible by the authorized users – application users and direct database users. To accomplish this, users need to execute a specific script that also uses Metadata DB to extract the list of Read-Only and Read-Write users for the specific application and specific environment. The access privileges may vary based on the environment. For example, Production and Production-like environments may have more restrictions on access whereas lower-level environments such as Dev and Test may have relaxed access. This different level of needs is captured and configured within the Metadata DB, which is then used by the tool. This tool first generates the SQL statements to assign privileges to the users and roles. Once the SQL script is generated for all required users and roles, then it gets executed.

3.4. Setup and Implementation

If we want to use the tool efficiently and effectively, then setting it up accurately is extremely crucial. Any mis-configuration can prevent the tool from being executed. The two main components of Setup and Implementation are: Metadata DB and Script locations on the server.

Metadata DB hosts all the common configuration tables such as environments, databases, servers, user credentials, dependent users, application types, current build numbers, master list of database changes per application, etc. All of this information and metadata is hosted in a database user named "TOOLS", which is like a master key for all the information. This user should not be allowed to be accessed by anyone else other than authorized users as this hosts the highly secured database user credentials information for all the environments, including Production. Since database objects are hosted within the users called owner accounts, which should only be accessed by administrators, who controls the changes to the database structures. No one else should be able to modify the structure of database objects.

Some of the Metadata DB tables are as follows:

ENVIRONMENTS: This table stores all the information about the environments, such as: application, current build#, environment type, owner credentials, read/read-write users' credentials, database, application roles (read, read-write).

HOSTS: This table stores the information about all the servers being used.

INSTANCES: This table stores data about all the database instances and on which servers they are hosted.

APP1_BUILDS: This is the master list table. It stores all the database changes for the application, APP1 in this case. If there are additional applications, specific tables would need to be created. Such as APP2_BUILDS, APP3_BUILDS etc. for applications APP2 and APP3, respectively. This table is the heart of the database change management system.

DEPENDENT_ENVIRONMENTS: This table stores the cross dependency between the environments. In a complex environment, one application may need to access the data in other applications or there may be several application users using the same application database schema. This table stores the interdependency and mark the dependencies as read-only or read-write. This table is the key for setting up the access privileges on the objects once the database changes are deployed.

The above tables are hosted in the Master configuration Metadata DB. There are other tables that are local to environments and stores environment specific metadata. These are the second piece of the puzzle which is solved when generating the pending changes list for any environment.

These tables are:

LAST_DB_CHANGE: This table is a single record table and always stores the latest database change build number. For the database change management system to proceed further, the build number in this table and the build number in the Environments table should match.

DB_CHANGE_RECORDS: This table stores all the changes implemented within an application database environment.

The above two tables play a key role when the database environment is refreshed from another database environment. These application-specific tables and the master configuration tables should be in sync for the solution to proceed further. This particular capability provides an extra verification mechanism before the system generates the executable for the pending changes. Accurate configuration of the above set of tables is a must for the database change management system to function without errors.

For the scripting portion of the solution, the scripts can be hosted on any Unix based server. The scrip for each application will have its dedicated home directory where the driver script will be hosted. Each database change build has a separate directory with a shell script and a SQL script. This is also hosted under the home directory of the application.

3.5. Software Replication

Replication simply refers to creating another copy exactly the same as the original one. This system allows to replicate itself and create copies that can be used for other applications. There are certain configuration steps that the users would need to follow to create a working copy of this solution.

These steps include creating the new application-specific configuration tables in the metadata database as discussed in the "Setup and Implementation" section, copying over the shell scripts to a new directory for the new application, replacing all the application-specific references from the original application to the new application and creating initial database build for application setup. As figure 6 indicates, there is a specific directory for application "app1", which hosts all the app1 related scripts. The database changes which are packaged into specific builds are hosted within this home directory. While replicating the solution, the entire app1 directory can be copied over except the underlying directories for app1 specific database changes.

3.6. On-Going Use, Maintenance, Enhancements

The command-line interface offers an easy to use option where authorized users can pass the application environment id and the target build number to the driver script. As explained in figure 7, the tool will then perform the preliminary verification steps and then generate an executable command as the output. The total number of pending builds for this environment and the individual builds will be listed. Once the tool completes execution, it syncs up the final build number in the master metadata tables and within the environment itself. This is to ensure that the next execution's initial check for build# verification passes through.

The tool will provide complete command after "EXECUTE COMMAND" as highlighted in red in Figure 7. This is the only command which needs to be executed to apply all the builds required up to the target build number. When the build is being executed, users can simultaneously monitor the execution with the "tail" command. Runtime monitoring is not necessary as the complete execution log will be stored in the log directory and can be reviewed later. If there are errors during the execution, an email with the errors will be sent to the configured distribution list.

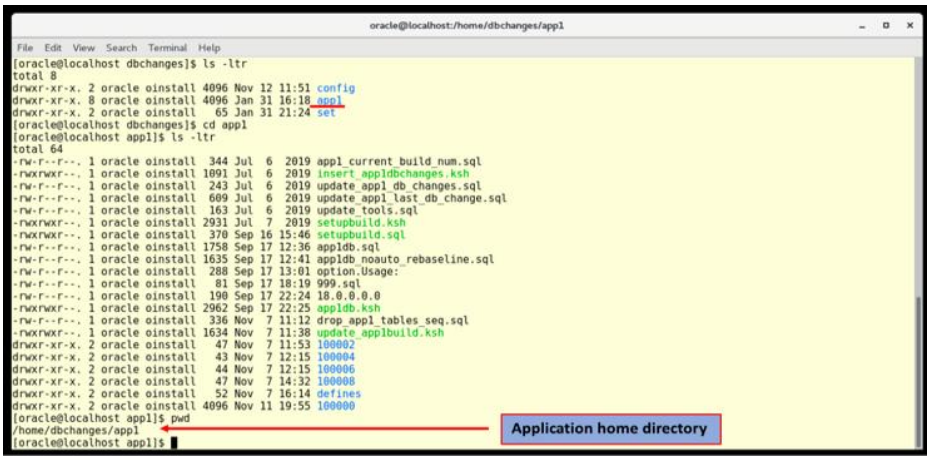

Figure 6. Application Directory for Replication

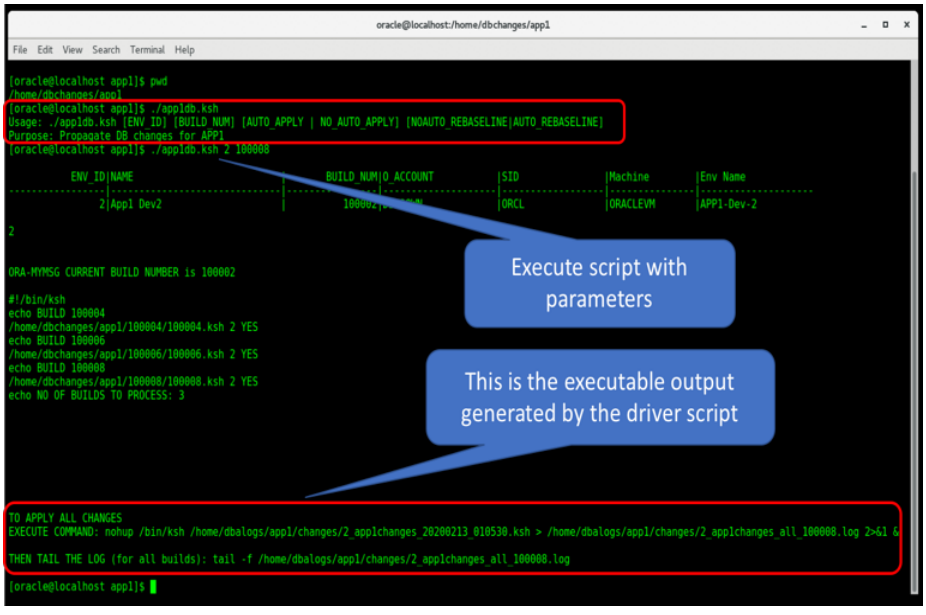

Figure 7. Executing the scripts

There are additional configuration options with this solution such as AUTO_APPLY and BASELINE. These options are available for configuring the solution when used with other software's within the organization. This allows the database change management system to be executed without manual intervention and also to be compatible with CI/CD tools. Organizations can leverage this functionality to scale up their software development speed and delivery.

Users also have the capability to disable a specific build by uncommenting the "exit" command which is included at the beginning of each build. This allows the user to seamlessly execute multiple builds and skip specific builds that needs any corrections or maintenance. The builds can be permanently disabled as well using this mechanism if the changes are no longer required for the release.

3.7. Benefits

This database change management system provides outstanding benefits which are top of the field in the area of database change management. The ***accuracy*** with which this system can deploy database changes across the various environments and releases makes it highly appealing for organizations doing software development. The change(s) will be applied to the specific application, for a specific release, and on a specific environment (Dev, Test, Performance, etc..).

The accuracy of implementation is coupled with detailed ***auditing*** functionality. Users can configure how long they would like to store the log files of execution, essentially allowing them to store logs indefinitely. These log files provide crucial metadata and execution sequence which is helpful when the teams are trying to debug and resolve any particular issue. Long term storage enables the organization with the data to go back even several years and reference, if needed. Logfiles of individual execution also provides the ability to ***compare*** the execution of same build across multiple environment.

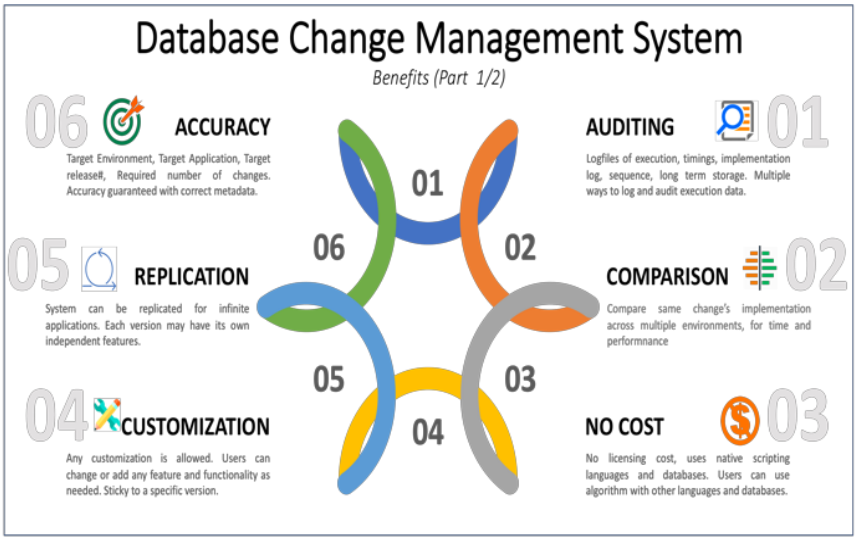

Figure 8. Benefits (1/2)

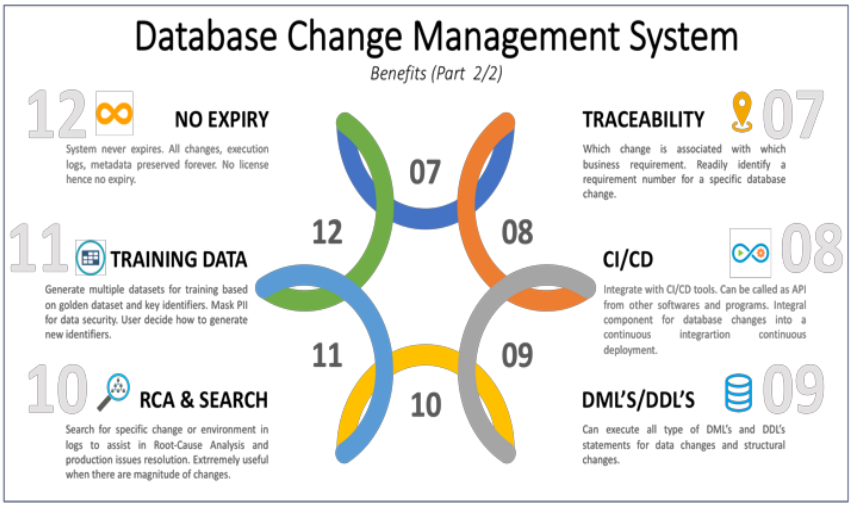

Figure 9. Benefits (2/2)

The solution guarantees that the requested build(s) will be applied to the intended environment and this environment only. This high confidence comes with ***no-cost*** associated. Generally, softwares have a licensing cost as well as e***nhancements*** and ***customization*** cost. Any change to the initially licensed version comes with a cost for the users and they have to completely depend on the software vendor for doing the customization and provide the new version. If the organization changes its mind in between or have another change, there is a cost associated to implement. This system comes with no licenses no cost. Anyone, individual or organizations, who can successfully implement the solution, can use it without paying a single penny for its licenses.

The solution can be ***replicated*** as and when needed without worrying about the cost as well as any enhancements or changes to specific versions can be done without incurring any cost. As far as the users understand the internal workings and logic of the system, they are free to make any changes, as per their requirements, for as many times as they want. Change made to one version of the system does not impact another version. For example, if the organization is using this system for two applications; APP1 and APP2, a customized change for APP1 has no impact on the code of the system for APP2 and its metadata. Each replicated version is completely independent from another having its own set of code and metadata.

Each change or database build within the database change management system is tagged with the business requirement number. This unique feature provides complete ***requirement traceability***. When there are hundreds of changes associated with a release, users can filter and search based on business requirement numbers to easily trace back the technical database change to specific business requirements. This feature is extremely helpful when debugging issues and performing root-cause analysis (RCA).

The solution can be used as an ***API*** and can be easily ***integrated*** with other tools used within the organization. This allows the organization to scale their ***CI/CD*** toolset by incorporating database change management in the integration and deployment process.

All types of SQL statements can be included in the builds. Where there are data changes, changes to database structures or applying referential integrity constraints. This solution is equally effective for ***DML's and DDL's***. This means that there is a single source of repository for all types of database changes for the organization, which is easily searchable based on key-words. This is a highly desirable feature when it comes to ***Root Cause Analysis (RCA)*** during critical production issues or resolving issues during development and test cycles.

The tool also provides a capability to generate production-like datasets to be used for ***training*** purposes. This plugin allows to configure which attributes are key identifiers, the logic to generate new identifiers, mask personally identifiable information (PII) for data security, and generate multiple datasets quickly. This way organization can replicate the data for as many trainees as required based on a given golden dataset.

The code for builds, driver scripts for the application(s), log files, metadata tables, application-specific changes never expires. This ***no-expiration*** benefit allows the organization to preserve and keep the whole suite of changes as long as they want. The scripts require very minimum storage and can be kept forever without any corruption concerns.

4. Applicable Industries and Domains

A solution or program is supplementary useful when more industries, companies working with varied domains of software development can use it. The value of such solutions increases exponentially. This database change management solution comes under the above category and is above and beyond useful for all types of software development. If the organization is involved in any type of software development that requires managing database changes with it, this solution stands out as the front runner, given the cost (which is zero), features, and its benefits to the organization.

The wide application of this solution is due to its flexibility of separating the driving scripts and how database changes are packaged for implementation. The database changes can belong to any type of industry, be it healthcare, insurance, automobile, transportation, aviation, integrated eligibility, manufacturing, research, or any other type. If there are database changes to be managed and the application uses a relational database management system, this system can be implemented. Start-up's as well as established large-scale organizations, this solution can help both of them with equally outstanding capabilities and performance.

5. Conclusions

Database changes are an integral part of the software development process and are foundational. Any organization involved in software development does database changes to implement the business requirements. These database changes can be industry-specific, business requirement specific, and/or required for regulatory compliance. Effectively storing the business data and be able to retrieve it within the SLAs is paramount to the success of a company. Such requirements call for a need for a solution that can manage the underlying data structures as well as keep up with accommodating and implementing the new changes.

This particular solution has the unique capability to be applicable for any industry or any domain and is not limited to a specific type of industry. From the very small-scale startup to a massively large organization managing only a few database changes to hundreds and thousands of database changes per release, this database change management system has worked with the same efficiency and performance regardless of the number of changes being processed through it.

This solution can serve uniformly well for public and private sector companies across the USA and around the

world. These organizations can be government organizations and agencies, healthcare, insurance, travel, maritime services, software consulting, transportation, data analytics & visualization, supply chain etc... The flexibility offered by the system to accommodate any industry-specific changes and still perform the database change management functionality renders it highly appealing to one and all.

REFERENCES

[1] Alex Yates, http://workingwithdevs.com/rolling-back-database-changes/, 2017.

[2] John Carbone, https://www.eetimes.com/timing-to-market-is-everything/#, EE Times, 2012.

[3] Steven Curtis,https://medium.com/swlh/environments-in-software-development-cf84adbbf197, Medium, 2020.

[4] Geoffrey Elliott, Global Business Information Technology: an integrated systems approach. Pearson Education. p.87., 2004.

[5] Managewell, http://managewell.net/?p=1157, 2011.

[6] Atlassian, https://www.atlassian.com/incident-management/kpis/cost-of-downtime, Gartner, 2016.

[7] M Sullivan, https://pubmed.ncbi.nlm.nih.gov/10119686/, 1991.

[8] Chris Doig, https://www.cio.com/article/2917796/why-comparing-enterprise-software-products-with-each-other-doesnt-identify-best-fit-software.html, 2015.

[9] Isaac Sacolick, https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html, 2020.

[10] David A. Garvin, https://hbr.org/1993/07/building-a-learning-organization, 1993.

[11] “The Definitive Glossary of Higher Mathematical Jargon — Algorithm". Math Vault. August 1, 2019.