[Responsibilities of an ETL tester](https://www.pavantestingtools.com/2014/08/responsibilities-of-etl-tester.html)

**[](https://4.bp.blogspot.com/-xG0KHwnrP3Q/XE64MOsfbhI/AAAAAAAAPzE/nuvzvwojlx4WJIAlmTWuIE6qzOkJK7tfwCLcBGAs/s1600/Programs%2Bfor%2BSelenium%25289%2529.png)**

**Responsibilities of an ETL tester**

Key responsibilities of an ETL tester are segregated into three categories

Stage table/ SFS or MFS

Business transformation logic applied

Target table loading from stage file or table after applying atransformation.

Some of the responsibilities of an ETL tester are

Test ETL software

Test components of  ETL datawarehouse

Execute backend data-driven test

Create, design and execute test cases, test plans and test harness

Identify the problem and provide solutions for potential issues

Approve requirements and design specifications

Data transfers and Test flat file

Writing SQL queries3 for various scenarios like count test

**ETL Performance Testing and Tuning**

ETL performance testingis a confirmation test to ensure that an ETL system can handle the load of multiple users and transactions.  The goal of performance tuning is to optimize session performance by eliminating performance bottlenecks. To tune or improve the performance of the session, you have to identify performance bottlenecks and eliminate it. Performance bottlenecks can be found in source and target databases, the mapping, the session and the system. One of the best tools used for performance testing is Informatica.

**Automation of ETL Testing**

The general methodology of ETL testing is to use SQL scripting or do “eyeballing” of data.. These approaches to ETL testing are time-consuming, error-prone and seldom provide complete test coverage. To accelerate, improve coverage, reduce costs, improve defect detection ration of ETL testing in production and development environments, automation is the need of the hour. One such tool is Informatica.

**Best Practices for ETL Testing**

Make sure data is transformed correctly

 Without any data loss and truncation projected data should be loaded into the data warehouse

 Ensure that ETL application appropriately rejects and replaces with default values and reports invalid data

 Need to ensure that the data loaded in data warehouse within prescribed and expected time frames to confirm scalability and performance

 All methods should have appropriate unit tests regardless of visibility

To measure their effectiveness all unit tests should use appropriate coverage techniques

Strive for one assertion per test case

 Create unit tests that target exceptions

|  |  |
| --- | --- |
| **ETL Testing** | **Data Base Testing** |
| Verifies whether data is moved as expected | The primary goal is to check if the data is following the rules/ standards defined in the Data Model |
| Verifies whether counts in the source and target are matching  Verifies whether the data transformed is as per expectation | Verify that there are no orphan records and foreign-primary key relations are maintained |
| Verifies that the foreign primary key relations are preserved during the ETL | Verifies that there are no redundant tables and database is optimally normalized |
| Verifies for duplication in loaded data | Verify if data is missing in columns where required |

[**ETL testing Fundamentals**](https://www.pavantestingtools.com/2015/05/etl-testing-fundamentals.html)

Comprehensive testing of a data warehouse at every point throughout the ETL (extract, transform, and load) process is becoming increasingly important as more data is being collected and used for strategic decision-making. Data warehouse or ETL testing is often initiated as a result of mergers and acquisitions, compliance and regulations, data consolidation, and the increased reliance on data-driven decision making (use of Business Intelligence tools, etc.). ETL testing is commonly implemented either manually or with the help of a tool (functional testing tool, ETL tool, proprietary utilities). Let us understand some of the basic ETL concepts.  
  
BI / Data Warehousing testing projects can be conjectured to be divided into ETL (Extract – Transform – Load) testing and henceforth the report testing.  
  
**Extract Transform Load** is the process to enable businesses to consolidate their data while moving it from place to place (i.e.) moving data from source systems into the data warehouse. The data can arrive from any source:  
  
  
**Extract -** It can be defined as extracting the data from numerous heterogeneous systems.  
  
**Transform -** Applying the business logics as specified b y the business on the data derived from sources.  
  
**Load -** Pumping the data into the final warehouse after completing the above two process. The ETL part of the testing mainly deals with how, when, from, where and what data we carry in our data warehouse from which the final reports are supposed to be generated. Thus, ETL testing spreads across all and each stage of data flow in the warehouse starting from the source databases to the final target warehouse.  
  
**Star Schema**  
The star schema is perhaps the simplest data warehouse schema. It is called a star schema because the entity-relationship diagram of this schema resembles a star, with points radiating from a central table. The center of the star consists of a large fact table and the points of the star are the dimension tables.  
A star schema is characterized by one OR more of very large fact tables that contain the primary information in the data warehouse, and a number of much smaller dimension tables (OR lookup tables), each of which contains information about the entries for a particular attribute in the fact table.  
  
A star query is a join between a fact table and a number of dimension tables. Each dimension table is joined to the fact table using a primary key to foreign key join, but the dimension tables are not joined to each other. The cost-based optimizer recognizes star queries and generates efficient execution plans for them. A typical fact table contains keys and measures. For example, in the sample schema, the fact table sales, contain the measures, quantity sold, amount, average, the keys time key, item-key, branch key, and location key. The dimension tables are time, branch, item and location.  
  
**Snow-Flake Schema**The snowflake schema is a more complex data warehouse model than a star schema, and is a type of star schema. It is called a snowflake schema because the diagram of the schema resembles a snowflake. Snowflake schemas normalize dimensions to eliminate redundancy. That is, the dimension data has been grouped into multiple tables instead of one large table.  
For example, a location dimension table in a star schema might be normalized into a location table and city table in a snowflake schema. While this saves space, it increases the number of dimension tables and requires more foreign key joins. The result is more complex queries and reduced query performance. Figure above presents a graphical representation of a snowflake schema.  
  
**When to use star schema and snowflake schema?**When we refer to Star and Snowflake Schemas, we are talking about a dimensional model for a Data Warehouse or a Datamart. The Star schema model gets it name from the design appearance because there is one central fact table surrounded by many dimension tables. The relationship between the fact and dimension tables is created by PK -> FK relationship and the keys are generally surrogate to the natural or business key of the dimension tables. All data for any given dimension is stored in the one dimension table. Thus, the design of the model could potentially look like a STAR. On the other hand, the Snowflake schema model breaks the dimension data into multiple tables for the purpose of making the data more easily understood or for reducing the width of the dimension table. An example of this type of schema might be a dimension with Product data of multiple levels. Each level in the Product Hierarchy might have multiple attributes that are meaningful only to that level. Thus, one would break the single dimension table into multiple tables in a hierarchical fashion with the highest level tied to the fact table. Each table in the dimension hierarchy would be tied to the level above by natural or business key where the highest level would be tied to the fact table by a surrogate key. As you can imagine the appearance of this schema design could resemble the appearance of a snowflake.  
  
**Types of Dimensions Tables**  
 **Type 1:** This is straightforward r e f r e s h . The fields are constantly overwritten and history is not kept for the column. For example should a description change for a Product number,the old value will be over written by the new value.  
**Type 2:** This is known as a slowly changing dimension, as history can be kept. The column(s) where the history is captured has to be defined. In our example of the Product description changing for a product number, if the slowly changing attribute captured is the product description, a new row of data will be created showing the new product description. The old description will still be contained in the old.  
**Type 3:** This is also a slowly changing dimension. However, instead of a new row, in the example, the old product description will be moved to an “old value” column in the dimension, while the new description will overwrite the existing column. In addition, a date stamp column exists to say when the value was updated. Although there will be no full history here, the previous value prior to the update is captured. No new rows will be created for history as the attribute is measured for the slowly changing value.  
  
**Types of fact tables:**  
 **Transactional:** Most facts will fall into this category. The transactional fact will capture transactional data such as sales lines or stock movement lines. The measures for these facts can be summed together.  
**Snapshot:** A snapshot fact will capture the current data for point for a day. For example, all the current stock positions, where items are, in which branch, at the end of a working day can be captured.  
Snapshot fact measures can be summed for this day, but cannot be summed across more than 2 snapshot days as this data will be incorrect.  
**Accumulative:**An accumulative snapshot will sum data up for an attribute, and is not based on time. For example, to get the accumulative sales quantity for a sale of a particular product, the row of data will be calculated for this row each night – giving an “accumulative” value.  
 **Key hit-points in ETL testing are:**There are several levels of testing that can be performed during data warehouse testing and they should be defined as part of the testing strategy in different phases (Component Assembly, Product) of testing. Some examples include:  
 **1. Constraint Testing:** During constraint testing, the objective is to validate unique constraints, primary keys, foreign keys, indexes, and relationships. The test script should include these validation points. Some ETL processes can be developed to validate constraints during the loading of the warehouse. If the decision is made to add constraint validation to the ETL process, the ETL code must validate all business rules and relational data requirements. In Automation, it should be ensured that the setup is done correctly and maintained throughout the ever-changing requirements process for effective testing. An alternative to automation is to use manual queries. Queries are written to cover all test scenarios and executed manually.  
 **2. Source to Target Counts:** The objective of the count test scripts is to determine if the record counts in the source match the record counts in the target. Some ETL processes are capable of capturing record count information such as records read, records written, records in error, etc. If the ETL process used can capture that level of detail and create a list of the counts, allow it to do so. This will save time during the validation process. It is always a good practice to use queries to double check the source to target counts.  
 **3. Source to Target Data Validation:**No ETL process is smart enough to perform source to target field-to-field validation. This piece of the testing cycle is the most labor intensive and requires the most thorough analysis of the data. There are a variety of tests that can be performed during source to target validation. Below is a list of tests that are best practices:  
 **4. Transformation and Business Rules:** Tests to verify all possible outcomes of the transformation rules, default values, straight moves and as specified in the Business Specification document. As a special mention, Boundary conditions must be tested on the business rules.  
 **5. Batch Sequence & Dependency Testing:** ETL’s in DW are essentially a sequence of processes that execute in a particular sequence. Dependencies do exist among various processes and the same is critical to maintain the integrity of the data. Executing the sequences in a wrong order might result in inaccurate data in the warehouse. The testing process must include at least 2 iterations of the end–end execution of the whole batch sequence. Data must be checked for its integrity during this testing. The most common type of errors caused because of incorrect sequence is the referential integrity failures, incorrect end-dating (if applicable) etc, reject  
records etc.  
 **6. Job restart Testing:**In a real production environment, the ETL jobs/processes fail because of number of reasons (say for ex: database related failures, connectivity failures etc). The jobs can fail half/partly executed. A good design always allows for a restart ability of the jobs from the failure point. Although this is more of a design suggestion/approach, it is suggested that every ETL job is built and tested for restart capability.  
 **7. Error Handling:** Understanding a script might fail during data validation, may confirm the ETL process is working through process validation. During process validation the testing team will work to identify additional data cleansing needs, as well as identify consistent error patterns that could possibly be diverted by modifying the ETL code. It is the responsibility of the validation team to identify any and all records that seem suspect. Once a record has been both data and process validated and the script has passed, the ETL process is functioning correctly. Conversely, if suspect records have been identified and documented during data validation those are not supported through process validation, the ETL process is not functioning correctly.  
 **8. Views:** Views created on the tables should be tested to ensure the attributes mentioned in the views are correct and the data loaded in the target table matches what is being reflected in the views.  
 **9. Sampling:** Sampling will involve creating predictions out of a representative portion of the data that is to be loaded into the target table; these predictions will be matched with the actual results obtained from the data loaded for business Analyst Testing. Comparison will be verified to ensure that the predictions match the data loaded into the target table.  
 **10. Process Testing:** The testing of intermediate files and processes to ensure the final outcome is valid and that performance meets the system/business need.  
 **11. Duplicate Testing:** Duplicate Testing must be performed at each stage of the ETL process and in the final target table. This testing involves checks for duplicates rows and also checks for multiple rows with same primary key, both of which cannot be allowed.  
 **12. Performance:** It is the most important aspect after data validation. Performance testing should check if the ETL process is completing within the load window.  
 **13. Volume:** Verify that the system can process the maximum expected quantity of data for a given cycle in the time expected.  
 **14.Connectivity Tests:** As the name suggests, this involves testing the upstream, downstream interfaces and intra DW connectivity. It is suggested that the testing represents the exact transactions between these interfaces. For ex: If the design approach is to extract the files from source system, we should actually test extracting a file out of the system and not just the  
connectivity.  
 **15. Negative Testing:** Negative Testing checks whether the application fails and where it should fail with invalid inputs and out of boundary scenarios and to check the behavior of the application.  
 **16. Operational Readiness Testing (ORT):** This is the final phase of testing which focuses on verifying the deployment of software and the operational readiness of the application. The main areas of testing in this phase include:  
  
Deployment Test  
  
  
1. Tests the deployment of the solution  
2. Tests overall technical deployment “checklist” and timeframes  
3. Tests the security aspects of the system including user authentication and  
authorization, and user-access levels.

|  |  |
| --- | --- |
| **Type of Bugs** | **Description** |
| User interface bugs/cosmetic bugs | ·          Related to GUI of application  ·          Font style, font size, colors, alignment, spelling mistakes, navigation and so on |
| Boundary Value Analysis (BVA) related bug | ·         Minimum and maximum values |
| Equivalence Class Partitioning (ECP) related bug | ·          Valid and invalid type |
| Input/Output bugs | ·         Valid values not accepted  ·          Invalid values accepted |
| Calculation bugs | ·         Mathematical errors  ·         Final output is wrong |
| Load Condition bugs | ·         Does not allows multiple users  ·         Does not allows customer expected load |
| Race Condition bugs | ·         System crash & hang  ·         System cannot run client platforms |
| Version control bugs | ·         No logo matching  ·          No version information available  ·         This occurs usually in regression testing |
| H/W bugs | ·         Device is not responding to the application |
| Help Source bugs | ·         Mistakes in help documents |

[**ETL Test Scenarios and Test Cases**](https://www.pavantestingtools.com/2014/08/etl-test-scenarios-and-test-cases.html)

|  |  |
| --- | --- |
| **Test Scenario** | **Test Cases**  **<https://2.bp.blogspot.com/-hdd0LzVtqlg/XE7MsiQkZTI/AAAAAAAAPzk/GRZ0c8DKZiMKOniESFPGbVZFIgm5-oregCEwYBhgL/s1600/Programs%2Bfor%2BSelenium%252812%2529.png>** |
| Mapping doc validation | Verify mapping doc whether corresponding ETL information is provided or not.  Change log should maintain in every mapping doc. |
| Validation | 1.      Validate the source and target table structure against corresponding mapping doc.  2.      Source data type and target data type should be same  3.      Length of data types in both source and target should be equal  4.      Verify that data field types and formats are specified  5.      Source data type length should not less than the target data type length  6.      Validate the name of columns in the table against mapping doc. |
| Constraint Validation | Ensure the constraints are defined for specific table as expected |
| Data consistency issues | 1.      The data type and length for a particular attribute may vary in files or tables though the semantic definition is the same.  2.      Misuse of integrity constraints |
| Completeness Issues | 1.      Ensure that all expected data is loaded into target table.  2.      Compare record counts between source and target.  3.      Check for any rejected records  4.      Check data should not be truncated in the column of target tables  5.      Check boundary value analysis  6.      Compares unique values of key fields between data loaded to WH and source data |
| Correctness Issues | 1.      Data that is misspelled or inaccurately recorded  2.      Null, non-unique or out of range data |
| Transformation | Transformation |
| Data Quality | 1.      Number check: Need to number check and validate it  2.      Date Check: They have to follow date format and it should be same across all records  3.      Precision Check  4.      Data check  5.      Null check |
| Null Validate | Verify the null values, where “Not Null” specified for a specific column. |
| Duplicate Check | 1.      Needs to validate the unique key, primary key and any other column should be unique as per the business requirements are having any duplicate rows  2.      Check if any duplicate values exist in any column which is extracting from multiple columns in source and combining into one column  3.      As per the client requirements, needs to be ensure that no duplicates in combination of multiple columns within target only |
| Date Validation | Date values are using many areas in ETL development for  1.      To know the row creation date  2.      Identify active records as per the ETL development perspective  3.      Identify active records as per the business requirements perspective  4.      Sometimes based on the date values the updates and inserts are generated. |
| Complete Data Validation | 1.      To validate the complete data set in source and target table minus a query in a best solution  2.      We need to source minus target and target minus source  3.      If minus query returns any value those should be considered as mismatching rows  4.      Needs to matching rows among source and target using intersect statement  5.      The count returned by intersect should match with individual counts of source and target tables  6.      If minus query returns of rows and count intersect is less than source count or target table then we can consider as duplicate rows are existed. |
| Data Cleanness | Unnecessary columns should be deleted before loading into the staging area. |

**How to create ETL Test Case**

ETL testing is a concept which can be applied to different tools and databases in information management industry.  **The objective of ETL testing is to assure that the data that has been loaded from a source to destination after business transformation is accurate.**  It also involves the verification of data at various middle stages that are being used between source and destination.

While performing ETL testing, two documents that will always be used by an ETL tester are

**ETL mapping sheets :**An ETL mapping sheets contain all the information of source and destination tables including each and every column and their look-up in reference tables. An ETL testers need to be comfortable with SQL queries as ETL testing may involve writing big queries with multiple joins to validate data at any stage of ETL. ETL mapping sheets provide a significant help while writing queries for data verification.

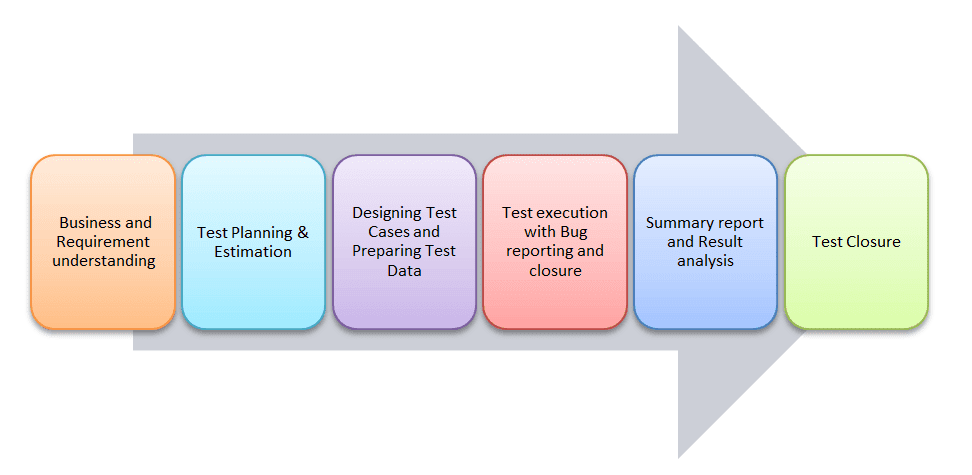
**DB Schema of Source, Target:**It should be kept handy to verify any detail in mapping sheets.

[**Types of ETL Testing**](https://www.pavantestingtools.com/2014/08/types-of-etl-testing.html)

|  |  |
| --- | --- |
| <https://2.bp.blogspot.com/-XZfY9i6iuV8/XFAV34QsyxI/AAAAAAAAP0E/MJtI9KQWkREGAP5sLxEl6a8BkQYni7GawCLcBGAs/s1600/Programs%2Bfor%2BSelenium%252814%2529.png>  **Types Of Testing** | **Testing Process** |
| Production Validation Testing | “Table balancing” or “production reconciliation” this type of ETL  testing is done on data as it is being moved into production systems.  To support your business decision, the data in your production systems has to be in the correct order.  Informatica Data Validation Option provides the ETL testing automation and management capabilities to ensure that production systems are not compromised by the data. |
| Source to Target Testing (Validation Testing) | Such type of testing is carried out to validate whether the data values transformed are the expected data values. |
| Application Upgrades | Such type of ETL testing can be automatically generated, saving substantial test development time. This type of testing checks whether the data extracted from an older application or repository are exactly same as the data in a repository or new application. |
| Metadata Testing | Metadata testing includes testing of data type check, data length check and index/constraint check. |
| Data Completeness Testing | To verify that all the expected data is loaded in target from the source, data completeness testing is done. Some of the tests that can be run are compare and validate counts, aggregates and actual data between the source and target for columns with simple transformation or no transformation. |
| Data Accuracy Testing | This testing is done to ensure that the data is accurately loaded and transformed as expected. |
| Data Transformation Testing | Testing data transformation is done as in many cases it cannot be achieved by writing one source SQL  query and comparing the output with the target.  Multiple SQL queries may need to be run for each row to verify the transformation rules. |
| Data Quality Testing | Data Quality Tests includes syntax and reference tests.  In order to avoid any error due to date or order number during business process Data Quality testing is done.  Syntax Tests: It will report dirty data,  based on invalid characters, character pattern, incorrect upper or lower case order etc.  Reference Tests: It will check the data according to the data model.  For example: Customer ID  Data quality testing includes number check, date check, precision check, data check , null check etc. |
| Incremental ETL testing | This testing is done to check the data integrity of old and new data with the addition of new data.  Incremental testing verifies that the inserts and updates are getting processed as expected during incremental ETL process. |
| GUI/Navigation Testing | This testing is done to check the navigation or GUI aspects of the front end reports. |

**ETL Testing Process**

Similar to other Testing Process, ETL also go through different phases. The different phases of ETL testing process is as follows



ETL testing is performed in five stages

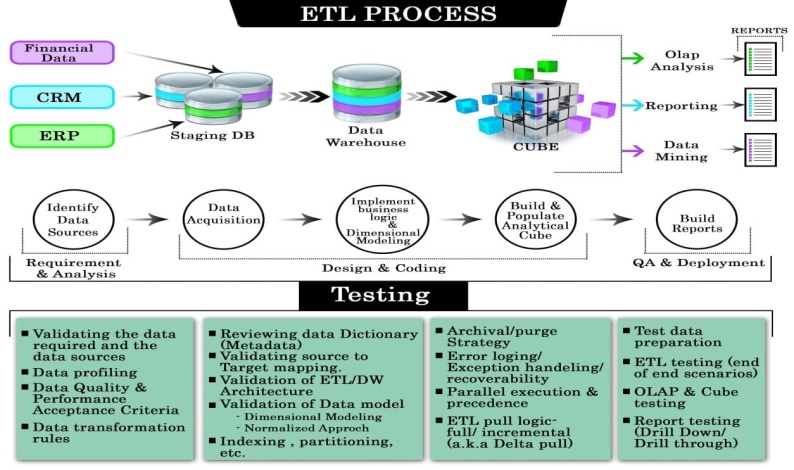
 Identifying data sources and requirements

Data acquisition

Implement business logics and dimensional Modelling

Build and populate data

Build Reports



[**Data Warehousing Concepts**](https://www.pavantestingtools.com/2014/02/data-warehousing-concepts.html)

[](https://2.bp.blogspot.com/-rNtLD8rugek/XFKpZyMEL9I/AAAAAAAAP5s/5SAz9eWYaNIIE42frUT6S6HKgjZNJlQ7wCLcBGAs/s1600/Programs%2Bfor%2BSelenium%25285%2529.png)

**Data Warehouse**  
-----------------------  
1) A data warehouse is a relational database that is designed for query and analysis rather than for transaction processing.  
2) A single, complete and consistent store of data obtained from a variety of different sources made available to end users in a format that they can understand and use in a business context.  
3) A technique for assembling and managing data from various sources for the purpose of answering business questions, thus making decisions that were previously not possible.  
4) It contains historical data derived from transaction data.  
5) It can include data from other sources.  
6) It is a consistent store of data obtained from a variety of different sources.  
7) It made available to end users in a way they can understand and use in a business context.  
  
**Why Data Warehouse and It’s testing?**  
  
Business Mandate: There are some major mergers and acquisition taking place and huge amount of data migration takes place  
Testing is to ensure that data transformation and data movement is correct.  
  
**Basic Elements of the Data Warehouse**  
-----------------------------------  
**Characteristics of Data ware House**  
Subject Oriented: Data warehouses are designed to help you analyze data.  
Integrated: Data warehouses must put data from disparate sources into a consistent format.  
They must resolve such problems as naming conflicts and inconsistencies among units of measure.  
Nonvolatile:  It means the data once entered into the data warehouse then it should not be changed.  
Time Variant: It means a data warehouse's focus on change over time.  
  
**Goals of a Data Warehouse**  
It must make an organization’s information more accessible  
It must make the organization’s information consistent  
It must be adaptive and resilient to change  
It must serve as a foundation for improved decision making  
  
**Data warehouse Advantages**  
To provide a consistent common source of information for various cross organizational and functional activity  
To Store Large Volumes of Historical Detail Data  
Improve the Ability to Access, Report Against, and Analyze Information  
To solve or improve upon Business Processes  
  
**Why do organizations need a data warehouse?**  
--------------------------------------------  
  
**Ad-hoc Reporting and Analysis**  
  
**Dynamic presentation through dashboards**  
  
**Predictive Analysis**  
 In business, predictive models utilize the patterns found in historical and transactional data to identify risks and opportunities.  
 Guiding in decision making for business expansion, business strategy etc.  
  
Predictive analytics is majorly used in  
  
 1)Financial services  
 2)Insurance  
 3)Retail  
4) Travel  
5)Healthcare  
6)Pharmaceuticals etc.  
  
  
**Different Schemas**  
--------------------  
Data Warehouse environment usually transforms the relational data model into some special architectures called **schema.**  
  
There are many schema models designed for data warehousing but the most commonly used are:  
  
1) Star schema  
2) Snowflake schema  
  
**There are 2 types of tables in data warehousing**  
  
**1) Fact Table**  
a Fact Table consists of the measurements, metrics or facts of a business process  
  
**2) Dimension Table**  
Dimension tables contain attributes that describe fact records in the fact table.  
  
  
**Warehouse Database - Types**  
----------------------------  
  
**Relational (ROLAP)**  
  
Central Warehouse is usually relational because of potentially large size of Data warehouse  
 **Multidimensional (MOLAP)**  
  
Faster response to analytical queries and OLAP computations but they have size limitations  
  
**Hybrid Architecture (HOLAP)**  
  
Uses relational component to support large databases and multidimensional component for fast response to analytical queries

[**ETL Testing / Data Warehouse Testing Overview**](https://www.pavantestingtools.com/2014/02/etl-testing-data-warehouse-testing.html)

ETL or Data warehouse testing is categorized into four different engagements irrespective of technology or ETL tools used:

• New Data Warehouse Testing – New DW is built and verified from scratch. Data input is taken from customer requirements and different data sources and new data warehouse is build and verified with the help of ETL tools.

• Migration Testing – In this type of project customer will have an existing DW and ETL performing the job but they are looking to bag new tool in order to improve efficiency.

• Change Request – In this type of project new data is added from different sources to an existing DW. Also, there might be a condition where customer needs to change their existing business rule or they might integrate the new rule.

• Report Testing – Report are the end result of any Data Warehouse and the basic propose for which DW is build. Report must be tested by validating layout, data in the report and calculation.

**ETL Testing Techniques:**

 1) Verify that data is transformed correctly according to various business requirements and rules.

 2) Make sure that all projected data is loaded into the data warehouse without any data loss and truncation.

 3) Make sure that ETL application appropriately rejects, replaces with default values and reports invalid data.

 4) Make sure that data is loaded in data warehouse within prescribed and expected time frames to confirm improved performance and scalability.

Apart from these 4 main ETL testing methods other testing methods like integration testing and user acceptance testing is also carried out to make sure everything is smooth and reliable.

**ETL Testing Process:**

Similar to any other testing that lies under Independent Verification and Validation, ETL also go through the same phase.

• Business and requirement understanding

• Validating

• Test Estimation

• Test planning based on the inputs from test estimation and business requirement

• Designing test cases and test scenarios from all the available inputs

• Once all the test cases are ready and are approved, testing team proceed to perform pre-execution check and test data preparation for testing

• Lastly execution is performed till exit criteria are met

• Upon successful completion summary report is prepared and closure process is done.

It is necessary to define test strategy which should be mutually accepted by stakeholders before starting actual testing. A well defined test strategy will make sure that correct approach has been followed meeting the testing aspiration. ETL testing might require writing SQL statements extensively by testing team or may be tailoring the SQL provided by development team. In any case testing team must be aware of the results they are trying to get using those SQL statements.

**Difference between Database and Data Warehouse Testing**

 There is a popular misunderstanding that database testing and data warehouse is similar while the fact is that both hold different direction in testing.

• Database testing is done using smaller scale of data normally with OLTP (Online transaction processing) type of databases while data warehouse testing is done with large volume with data involving OLAP (online analytical processing) databases.

• In database testing normally data is consistently injected from uniform sources while in data warehouse testing most of the data comes from different kind of data sources which are sequentially inconsistent.

• We generally perform only CRUD (Create, read, update and delete) operation in database testing while in data warehouse testing we use read-only (Select) operation.

• Normalized databases are used in DB testing while demoralized DB is used in data warehouse testing.

There are number of universal verifications that have to be carried out for any kind of data warehouse testing.  
Below is the list of objects that are treated as essential for validation in ETL testing:

 - Verify that data transformation from source to destination works as expected

 - Verify that expected data is added in target system

 - Verify that all DB fields and field data is loaded without any truncation

 - Verify data checksum for record count match

 - Verify that for rejected data proper error logs are generated with all details

 - Verify NULL value fields

 - Verify that duplicate data is not loaded

 - Verify data integrity

**ETL Testing Challenges:**

ETL testing is quite different from conventional testing. There are many challenges we faced while performing data warehouse testing. Here is the list of few ETL testing challenges I experienced on my project:

 - Incompatible and duplicate data.

 - Loss of data during ETL process.

 - Unavailability of inclusive test bed.

 - Testers have no privileges to execute ETL jobs by their own.

 - Volume and complexity of data is very huge.

 - Fault in business process and procedures.

 - Trouble acquiring and building test data.

 - Missing business flow information.

Data is important for businesses to make the critical business decisions. ETL testing plays a significant role validating and ensuring that the business information is exact, consistent and reliable. Also, it minimizes hazard of data loss in production.

Source\_schema validation:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test case #** | **Test Case Name** | **Test Case Type** | **Step No** | **Action/Query** | **Expected Results** | **Comments/Queries** | **Status** |
| **TC\_HRSRC\_EMP01** | **Check number of columns in EMPLOYEES Table** | Source Schema validation | 1 | select count(\*)  from user\_tab\_columns where table\_name='EMPLOYEES'; | 11 | 11 | Passed |
| **TC\_HRSRC\_EMP02** | **Check Data type of the columns in EMPLOYEES Table** | Source Schema validation | 1 | SELECT column\_name, data\_type FROM user\_tab\_columns where table\_name = 'EMPLOYEES'; | Verify result should be as per the design. | as expected | Passed |
| **TC\_HRSRC\_EMP03** | **Check size of the columns in EMPLOYEES Table** | Source Schema validation | 1 | SELECT column\_name, data\_length FROM user\_tab\_columns where table\_name = 'EMPLOYEES'; | Verify result should be as per the design. |  |  |
| **TC\_HRSRC\_EMP04** | **Check Contraints of columns in EMPLOYEES Table** | Source Schema validation | 1 | SELECT column\_name,constraint\_name from user\_cons\_columns where table\_name = 'EMPLOYEES'; | Verify result should be as per the design. | as expected | Passed |
| **TC\_HRSRC\_EMP05** | **Check Indexes of Columns in EMPLOYEES Table** | Source Schema validation | 1 | SELECT COLUMN\_NAME,INDEX\_NAME from dba\_ind\_columns where table\_name='EMPLOYEES' AND INDEX\_OWNER='HR'; | Verify result should be as per the design. | as expected | Passed |

Target shema validation:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test case #** | **Test Case Name** | **Test Case Type** | **Step No** | **Action/Query** | **Expected Results** | **Comments/Queries** | **Result** |
| **TC\_HRTRG\_EMP01** | Check number of columns in EMP\_TOTSAL\_TRG Table | Target schema validation | 1 | select count(\*)  from user\_tab\_columns where table\_name='EMP\_TOTSAL\_TRG'; | 10 | As Expecteds | Passed |
| **TC\_HRTRG\_EMP02** | Check Data type of the columns in EMP\_TOTSAL\_TRG Table | Target schema validation | 1 | SELECT column\_name, data\_type FROM user\_tab\_columns where table\_name = 'EMP\_TOTSAL\_TRG'; | Verify result should be as per the design. |  |  |
| **TC\_HRTRG\_EMP03** | Check size of the columns in EMP\_TOTSAL\_TRG Table | Target schema validation | 1 | SELECT column\_name, data\_length FROM user\_tab\_columns where table\_name = 'EMP\_TOTSAL\_TRG'; | Verify result should be as per the design. |  |  |
| **TC\_HRTRG\_EMP04** | Check Contraints of columns in EMP\_TOTSAL\_TRG Table | Target schema validation | 1 | SELECT column\_name,constraint\_name from user\_cons\_columns where table\_name = 'EMP\_TOTSAL\_TRG'; | Verify result should be as per the design. | As Expecteds | Passed |
| **TC\_HRTRG\_EMP05** | Check Indexes of Columns in EMP\_TOTSAL\_TRG Table | Target schema validation | 1 | SELECT COLUMN\_NAME,INDEX\_NAME from dba\_ind\_columns where table\_name='EMP\_TOTSAL\_TRG' AND INDEX\_OWNER='HR'; | Verify result should be as per the design. | indexes not mapped | Failed |

Mapping Test cases:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test case #** | **Test Case Name** | **Test Case Type** | **Step No** | **Action/Query** | **Expected Results** | **Actual Results** |
| **TC\_MAP1\_01** | Check number of records present in Source table | Data validation | 1 | Select count(\*) From EMPLOYEES | 107 | Passed |
| **TC\_MAP1\_02** | Check number of records present in Target table after data is loaded | Data validation | 2 | Select count(\*) From EMP\_TOTSAL\_TRG | 107 | Passed |
| **TC\_MAP1\_03** | Check the records pulled from source table(s) after tranformations applied | Data validation | 3 | SELECT EMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER , ROUND((SYSDATE-HIRE\_DATE)/365,0) EXPEREINCE, JOB\_ID,  NVL(SALARY,0) + NVL(COMMISSION\_PCT,0) TOTAL\_SALARY,  MANAGER\_ID,DEPARTMENT\_ID FROM EMPLOYEES | As per the design | Passed |
| **TC\_MAP1\_04** | Check the records loaded in to target table(s) | Data validation | 4 | SELECTEMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER ,EXPEREINCE,TOTAL\_SALARY,MANAGER\_ID,DEPARTMENT\_ID FROM EMP\_TOTSAL\_TRG | As per the design | Passed |
| **TC\_MAP1\_05** | Check the records present in source table which are not in target table | Data validation | 5 | SELECT EMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER , ROUND((SYSDATE-HIRE\_DATE)/365,0) EXPEREINCE, JOB\_ID,  NVL(SALARY,0) + NVL(COMMISSION\_PCT,0) TOTAL\_SALARY,  MANAGER\_ID,DEPARTMENT\_ID FROM EMPLOYEES  MINUS  SELECTEMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER ,EXPEREINCE,TOTAL\_SALARY,MANAGER\_ID,DEPARTMENT\_ID FROM EMP\_TOTSAL\_TRG | No Records Found | Failed |
| **TC\_MAP1\_06** | Check the records present in target table which are not in source table | Data validation | 6 | SELECTEMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER ,EXPEREINCE,TOTAL\_SALARY,MANAGER\_ID,DEPARTMENT\_ID FROM EMP\_TOTSAL\_TRG  MINUS  SELECT EMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER , ROUND((SYSDATE-HIRE\_DATE)/365,0) EXPEREINCE, JOB\_ID,  NVL(SALARY,0) + NVL(COMMISSION\_PCT,0) TOTAL\_SALARY,  MANAGER\_ID,DEPARTMENT\_ID FROM EMPLOYEES; | No Records Found | Failed |
| **TC\_MAP1\_07** | Check if any records missed from source to tartget tables | Data validation | 7 | SELECT EMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER , ROUND((SYSDATE-HIRE\_DATE)/365,0) EXPEREINCE, JOB\_ID,  NVL(SALARY,0) + NVL(COMMISSION\_PCT,0) TOTAL\_SALARY,  MANAGER\_ID,DEPARTMENT\_ID FROM EMPLOYEES  MINUS  SELECTEMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER ,EXPEREINCE,TOTAL\_SALARY,MANAGER\_ID,DEPARTMENT\_ID FROM EMP\_TOTSAL\_TRG  UNION  SELECTEMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER ,EXPEREINCE,TOTAL\_SALARY,MANAGER\_ID,DEPARTMENT\_ID FROM EMP\_TOTSAL\_TRG  MINUS  SELECT EMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER , ROUND((SYSDATE-HIRE\_DATE)/365,0) EXPEREINCE, JOB\_ID,  NVL(SALARY,0) + NVL(COMMISSION\_PCT,0) TOTAL\_SALARY,  MANAGER\_ID,DEPARTMENT\_ID FROM EMPLOYEES; | No Records Found | Failed |
| **TC\_MAP1\_08** | Check if any Duplicate records in target tables | Data validation | 8 | SELECT \* FROM EMP\_TOTSAL\_TRG WHERE ROWID IN ( SELECT rowid FROM EMP\_TOTSAL\_TRG GROUP BY rowid HAVING COUNT(\*)>1); | No Records Found |  |

ETL TEST RESULTS:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test case #** | **Test Case Name** | **Test Case Type** | **Step No** | **Action/Query** | **Expected Results** | **Test Results** |
| **TC\_HRSRC\_EMP01** | Check number of columns in EMPLOYEES Table | Source Schema validation | 1 | select count(\*)  from user\_tab\_columns where table\_name='EMPLOYEES'; | 11 |  |
| **TC\_HRSRC\_EMP02** | Check Data type of the columns in EMPLOYEES Table | Source Schema validation | 1 | SELECT column\_name, data\_type FROM user\_tab\_columns where table\_name = 'EMPLOYEES'; | Verify result should be as per the design. |  |
| **TC\_HRSRC\_EMP03** | Check size of the columns in EMPLOYEES Table | Source Schema validation | 1 | SELECT column\_name, data\_length FROM user\_tab\_columns where table\_name = 'EMPLOYEES'; | Verify result should be as per the design. |  |
| **TC\_HRSRC\_EMP04** | Check Contraints of columns in EMPLOYEES Table | Source Schema validation | 1 | SELECT column\_name,constraint\_name from user\_cons\_columns where table\_name = 'EMPLOYEES'; | Verify result should be as per the design. |  |
| **TC\_HRSRC\_EMP05** | Check Indexes of Columns in EMPLOYEES Table | Source Schema validation | 1 | SELECT COLUMN\_NAME,INDEX\_NAME from dba\_ind\_columns where table\_name='EMPLOYEES' AND INDEX\_OWNER='HR'; | Verify result should be as per the design. |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **TC\_HRTRG\_EMP01** | Check number of columns in EMP\_TOTSAL\_TRG Table | Target schema validation | 1 | select count(\*)  from user\_tab\_columns where table\_name='EMP\_TOTSAL\_TRG'; | 10 |  |
| **TC\_HRTRG\_EMP02** | Check Data type of the columns in EMP\_TOTSAL\_TRG Table | Target schema validation | 1 | SELECT column\_name, data\_type FROM user\_tab\_columns where table\_name = 'EMP\_TOTSAL\_TRG'; | Verify result should be as per the design. |  |
| **TC\_HRTRG\_EMP03** | Check size of the columns in EMP\_TOTSAL\_TRG Table | Target schema validation | 1 | SELECT column\_name, data\_length FROM user\_tab\_columns where table\_name = 'EMP\_TOTSAL\_TRG'; | Verify result should be as per the design. |  |
| **TC\_HRTRG\_EMP04** | Check Contraints of columns in EMP\_TOTSAL\_TRG Table | Target schema validation | 1 | SELECT column\_name,constraint\_name from user\_cons\_columns where table\_name = 'EMP\_TOTSAL\_TRG'; | Verify result should be as per the design. |  |
| **TC\_HRTRG\_EMP05** | Check Indexes of Columns in EMP\_TOTSAL\_TRG Table | Target schema validation | 1 | SELECT COLUMN\_NAME,INDEX\_NAME from dba\_ind\_columns where table\_name='EMP\_TOTSAL\_TRG' AND INDEX\_OWNER='HR'; | Verify result should be as per the design. |  |
|  |  |  |  |  |  |  |
| **TC\_MAP1\_01** | Check number of records present in Source table | Data validation | 1 | Select count(\*) From EMPLOYEES | 11 |  |
| **TC\_MAP1\_02** | Check number of records present in Target table after data is loaded | Data validation | 2 | Select count(\*) From EMP\_TOTSAL\_TRG | 11 |  |
| **TC\_MAP1\_03** | Check the records pulled from source table(s) after tranformations applied | Data validation | 3 | SELECT EMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER , ROUND((SYSDATE-HIRE\_DATE)/365,0) EXPEREINCE, JOB\_ID,  NVL(SALARY,0) + NVL(COMMISSION\_PCT,0) TOTAL\_SALARY,  MANAGER\_ID,DEPARTMENT\_ID FROM EMPLOYEES | As per the design |  |
| **TC\_MAP1\_04** | Check the records loaded in to target table(s) | Data validation | 4 | SELECTEMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER ,EXPEREINCE,TOTAL\_SALARY,MANAGER\_ID,DEPARTMENT\_ID FROM EMP\_TOTSAL\_TRG | As per the design |  |
| **TC\_MAP1\_05** | Check the records present in source table which are not in target table | Data validation | 5 | SELECT EMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER , ROUND((SYSDATE-HIRE\_DATE)/365,0) EXPEREINCE, JOB\_ID,  NVL(SALARY,0) + NVL(COMMISSION\_PCT,0) TOTAL\_SALARY,  MANAGER\_ID,DEPARTMENT\_ID FROM EMPLOYEES  MINUS  SELECTEMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER ,EXPEREINCE,TOTAL\_SALARY,MANAGER\_ID,DEPARTMENT\_ID FROM EMP\_TOTSAL\_TRG | No Records Found |  |
| **TC\_MAP1\_06** | Check the records present in target table which are not in source table | Data validation | 6 | SELECTEMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER ,EXPEREINCE,TOTAL\_SALARY,MANAGER\_ID,DEPARTMENT\_ID FROM EMP\_TOTSAL\_TRG  MINUS  SELECT EMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER , ROUND((SYSDATE-HIRE\_DATE)/365,0) EXPEREINCE, JOB\_ID,  NVL(SALARY,0) + NVL(COMMISSION\_PCT,0) TOTAL\_SALARY,  MANAGER\_ID,DEPARTMENT\_ID FROM EMPLOYEES; | No Records Found |  |
| **TC\_MAP1\_07** | Check if any records missed from source to tartget tables | Data validation | 7 | SELECT EMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER , ROUND((SYSDATE-HIRE\_DATE)/365,0) EXPEREINCE, JOB\_ID,  NVL(SALARY,0) + NVL(COMMISSION\_PCT,0) TOTAL\_SALARY,  MANAGER\_ID,DEPARTMENT\_ID FROM EMPLOYEES  MINUS  SELECTEMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER ,EXPEREINCE,TOTAL\_SALARY,MANAGER\_ID,DEPARTMENT\_ID FROM EMP\_TOTSAL\_TRG  UNION  SELECTEMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER ,EXPEREINCE,TOTAL\_SALARY,MANAGER\_ID,DEPARTMENT\_ID FROM EMP\_TOTSAL\_TRG  MINUS  SELECT EMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER , ROUND((SYSDATE-HIRE\_DATE)/365,0) EXPEREINCE, JOB\_ID,  NVL(SALARY,0) + NVL(COMMISSION\_PCT,0) TOTAL\_SALARY,  MANAGER\_ID,DEPARTMENT\_ID FROM EMPLOYEES; | No Records Found |  |
| **TC\_MAP1\_08** | Check if any Duplicate records in target tables | Data validation | 8 | SELECT \* FROM EMP\_TOTSAL\_TRG WHERE ROWID IN ( SELECT rowid FROM EMP\_TOTSAL\_TRG GROUP BY rowid HAVING COUNT(\*)>1); | No Records Found |  |

BUG REPORTS:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SNO** | **ID** | **TITLE** | **PRIORITY** | **SEVIROTY** | **DESCRIPTION** | **STEPS** | **CATEGORY** | **REPORTED BY** | **ASSIGNED TO** | **STATUS** | **RESOLUTION TYPE** | **BUILDNO** | **REGRESSION(Y/N)** |
| 1 | HR\_ETL\_1 | Indexes are not mapped to target table **EMP\_TOTSAL\_TRG** | 2 | 2 | Indexes are not mapped to target table EMP\_TOTSAL\_TRG | SELECT COLUMN\_NAME,INDEX\_NAME from dba\_ind\_columns where table\_name='EMP\_TOTSAL\_TRG' AND INDEX\_OWNER='HR'; | sanity | xxx | xxx | new | xxxxx | 1 | N |
| 2 | HR\_ETL\_2 | Data is incorrect in target tables for Mapping-1 | 1 | 1 | Data is incorrect in target tables for Mapping-1 | SELECT EMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER , ROUND((SYSDATE-HIRE\_DATE)/365,0) EXPEREINCE, JOB\_ID,  NVL(SALARY,0) + NVL(COMMISSION\_PCT,0) TOTAL\_SALARY,  MANAGER\_ID,DEPARTMENT\_ID FROM EMPLOYEES  MINUS  SELECTEMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER ,EXPEREINCE,TOTAL\_SALARY,MANAGER\_ID,DEPARTMENT\_ID FROM EMP\_TOTSAL\_TRG  UNION  SELECTEMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER ,EXPEREINCE,TOTAL\_SALARY,MANAGER\_ID,DEPARTMENT\_ID FROM EMP\_TOTSAL\_TRG  MINUS  SELECT EMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER , ROUND((SYSDATE-HIRE\_DATE)/365,0) EXPEREINCE, JOB\_ID,  NVL(SALARY,0) + NVL(COMMISSION\_PCT,0) TOTAL\_SALARY,  MANAGER\_ID,DEPARTMENT\_ID FROM EMPLOYEES; | Functional |  |  |  |  | 1 | N |
| 3 | HR\_ETL\_3 |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 | HR\_ETL\_4 |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 | HR\_ETL\_5 |  |  |  |  |  |  |  |  |  |  |  |  |

Source Table:

|  |  |  |  |
| --- | --- | --- | --- |
| **TABLE-1** | **COLUMN NAMES** | **DATA TYPES** | **KEYS** |
| **EMPLOYEES** | EMPLOYEE\_ID | NUMBER(6) | **P** |
| FIRST\_NAME | VARCHAR2(20) |  |
| LAST\_NAME | VARCHAR2(20) |  |
| EMAIL | VARCHAR2(25) | **U** |
| PHONE\_NUMBER | VARCHAR2(20) |  |
| HIRE\_DATE | DATE |  |
| JOB\_ID | VARCHAR2(10) | **F** |
| SALARY | NUMBER(8,2) |  |
| COMMISSION\_PCT | NUMBER(2,2) |  |
| MANAGER\_ID | NUMBER(6) | **F** |
| DEPARTMENT\_ID | NUMBER(4) | **F** |
| **TABLE-2** | **COLUMN NAMES** | **DATA TYPES** | **KEYS** |
| **DEPARTMENTS** | DEPARTMENT\_ID | NUMBER(4) | **P** |
| DEPARTMENT\_NAME | VARCHAR2(30) |  |
| MANAGER\_ID | NUMBER(6) | **F** |
| LOCATION\_ID | NUMBER(4) | **F** |
| **TABLE-3** | **COLUMN NAMES** | **DATA TYPES** | **KEYS** |
| **JOBS** | JOB\_ID | VARCHAR2(10) | **P** |
| JOB\_TITLE | VARCHAR2(35) |  |
| MIN\_SALARY | NUMBER(6) |  |
| MAX\_SALARY | NUMBER(6) |  |
| **TABLE-4** | **COLUMN NAMES** | **DATA TYPES** | **KEYS** |
| **JOB\_HISTORY** | EMPLOYEE\_ID | NUMBER(6) | **PF** |
| START\_DATE | DATE | **P** |
| END\_DATE | DATE |  |
| JOB\_ID | VARCHAR2(10) | **F** |
| DEPARTMENT\_ID | NUMBER(4) | **F** |
| **TABLE-5** | **COLUMN NAMES** | **DATA TYPES** | **KEYS** |
| **REGIONS** | REGION\_ID | NUMBER | P |
| REGION\_NAME | VARCHAR2(25) |  |
| **TABLE-6** | **COLUMN NAMES** | **DATA TYPES** | **KEYS** |
| **COUNTRIES** | COUNTRY\_ID | CHAR(2) | P |
| COUNTRY\_NAME | VARCHAR2(40) |  |
| REGION\_ID | NUMBER |  |
|  |  |  |  |
|  |  |  |  |
| **TABLE-7** | **COLUMN NAMES** | **DATA TYPES** | **KEYS** |
| **LOCATIONS** | LOCATION\_ID | NUMBER(4) | P |
| STREET\_ADDRESS | VARCHAR2(40) |  |
| POSTAL\_CODE | VARCHAR2(12) |  |
| CITY | VARCHAR2(30) |  |
| STATE\_PROVINCE | VARCHAR2(25) |  |
| COUNTRY\_ID | CHAR(2) | F |

Target Table:

|  |  |  |
| --- | --- | --- |
| **TABLE-1** | **COLUMN NAMES** | **DATA TYPES** |
| **EMP\_TOTSAL\_TRG** | EMPLOYEE\_ID | NUMBER(6) |
| FIRST\_NAME | VARCHAR2(20) |
| LAST\_NAME | VARCHAR2(20) |
| EMAIL | VARCHAR2(25) |
| PHONE\_NUMBER | VARCHAR2(20) |
| EXPERIENCE | NUMBER(2) |
| JOB\_ID | VARCHAR2(10) |
| TOTAL\_SALARY | NUMBER(8,2) |
| MANAGER\_ID | NUMBER(6) |
| DEPARTMENT\_ID | NUMBER(4) |
|  |  |  |
|  |  |  |
| **TABLE-2** | **COLUMN NAMES** | **DATA TYPES** |
| **EMP\_DEPT\_TRG** | EMPLOYEE\_ID | NUMBER(6) |
| FIRST\_NAME | VARCHAR2(20) |
| LAST\_NAME | VARCHAR2(20) |
| EMAIL | VARCHAR2(25) |
| PHONE\_NUMBER | VARCHAR2(20) |
| HIRE\_DATE | DATE |
| JOB\_ID | VARCHAR2(10) |
| SALARY | NUMBER(8,2) |
| COMMISSION\_PCT | NUMBER(2,2) |
| MANAGER\_ID | NUMBER(6) |
| DEPARTMENT\_ID | NUMBER(4) |
|  |  |  |
|  |  |  |
| **TABLE-3** | **COLUMN NAMES** | **DATA TYPES** |
| **EMP\_SUMSAL\_DEPTWISE\_TRG** | DEPARTMENT\_ID | NUMBER(4) |
| SUM\_SALARY | NUMBER(5) |
|  |  |  |
| **TABLE-4** | **COLUMN NAMES** | **DATA TYPES** |
| **EMP\_DEPTNAME\_TRG** | EMPLOYEE\_ID | NUMBER(6) |
| FIRST\_NAME | VARCHAR2(20) |
| LAST\_NAME | VARCHAR2(20) |
| EMAIL | VARCHAR2(25) |
| PHONE\_NUMBER | VARCHAR2(20) |
| HIRE\_DATE | DATE |
| JOB\_ID | VARCHAR2(10) |
| SALARY | NUMBER(8,2) |
| COMMISSION\_PCT | NUMBER(2,2) |
| MANAGER\_ID | NUMBER(6) |
| DEPARTMENT\_ID | NUMBER(4) |
| DEPARTMENT\_NAME | VARCHAR2(30) |
|  |  |  |
|  |  |  |
| **TABLE-5** | **COLUMN NAMES** | **DATA TYPES** |
| **EMP\_COUNT\_SAL\_DEPWISE\_TRG** | DEPARTMENT\_ID | NUMBER(4) |
| NUM\_OF\_EMPLOYEES | NUMBER(5) |
| MAXSAL | NUMBER(5,5) |
| MINSAL | NUMBER(5,5) |
| AVGSAL | NUMBER(5,5) |
|  |  |  |
| **TABLE-6** | **COLUMN NAMES** | **DATA TYPES** |
| **EMP\_MANAGER\_TRG** | EMPLOYEE\_ID | NUMBER(4) |
| FIRST\_NAME | VARCHAR2(10) |
| MANAGER\_NAME | VARCHAR2(10) |
|  |  |  |
|  |  |  |
| **TABLE-7** | **COLUMN NAMES** | **DATA TYPES** |
| **EMP\_SAL\_RANKS\_TRG** | EMPLOYEE\_ID | NUMBER(4) |
| FIRST\_NAME | VARCHAR2(10) |
| SALARY | NUMBER(8,2) |
| RANK | NUMBER(2) |
| DENSE\_RANK | NUMBER(2) |
| ROWNUMBER | NUMBER(2) |
|  |  |  |

Mapping Sheets:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MAPPING-1** |  |  |  |  |
|  |  |  |  |  |
| **SOURCE** | | **TRANSFORMATIONS** | **TARGET** | |
| **HR.EMPLOYEES** | |  | **HR.EMP\_TOTSAL\_TRG** | |
| **Column Names** | **Data Types** | **Expression Tranformation** | **Column Names** | **Data Types** |
| EMPLOYEE\_ID | NUMBER(6) | Direct Move | EMPLOYEE\_ID | NUMBER(6) |
| FIRST\_NAME | VARCHAR2(20) | Direct Move | FIRST\_NAME | VARCHAR2(20) |
| LAST\_NAME | VARCHAR2(20) | Direct Move | LAST\_NAME | VARCHAR2(20) |
| EMAIL | VARCHAR2(25) | Direct Move | EMAIL | VARCHAR2(25) |
| PHONE\_NUMBER | VARCHAR2(20) | Direct Move | PHONE\_NUMBER | VARCHAR2(20) |
| HIRE\_DATE | DATE | ***round((sysdate-Hiredate)/365,0)*** | EXPERIENCE | NUMBER(2) |
| JOB\_ID | VARCHAR2(10) | Direct Move | JOB\_ID | VARCHAR2(10) |
| SALARY | NUMBER(8,2) | ***Sal+Comm; Check NULL values in each column And if NULL value is encountered overwrite it to Zero.*** | TOTAL\_SALARY | NUMBER(8,2) |
| COMMISSION\_PCT | NUMBER(2,2) |
| MANAGER\_ID | NUMBER(6) | Direct Move | MANAGER\_ID | NUMBER(6) |
| DEPARTMENT\_ID | NUMBER(4) | Direct Move | DEPARTMENT\_ID | NUMBER(4) |
|  |  |  |  |  |
|  |  |  |  |  |
| **MAPPING-2** |  |  |  |  |
|  |  |  |  |  |
| **SOURCE** | | **TRANSFORMATIONS** | **TARGET** | |
| **HR.EMPLOYEES** | |  | **HR.SUM\_SAL\_DEPWISE\_TRG** | |
| **Column Names** | **Data Types** | **Aggregator Transformation** | **Column Names** | **Data Types** |
| EMPLOYEE\_ID | NUMBER(6) | Don’t move |  |  |
| FIRST\_NAME | VARCHAR2(20) | Don’t move |  |  |
| LAST\_NAME | VARCHAR2(20) | Don’t move |  |  |
| EMAIL | VARCHAR2(25) | Don’t move |  |  |
| PHONE\_NUMBER | VARCHAR2(20) | Don’t move |  |  |
| HIRE\_DATE | DATE | Don’t move |  |  |
| JOB\_ID | VARCHAR2(10) | Don’t move |  |  |
| SALARY | NUMBER(8,2) | ***sum(sal) department wise*** | SUM\_SALARY | NUMBER(5) |
| COMMISSION\_PCT | NUMBER(2,2) | Don’t move |  |  |
| MANAGER\_ID | NUMBER(6) | Don’t move |  |  |
| DEPARTMENT\_ID | NUMBER(4) | Direct move | DEPARTMENT\_ID | NUMBER(4) |
|  |  |  |  |  |
|  |  |  |  |  |
| **MAPPING-3** |  |  |  |  |
|  |  |  |  |  |
| **SOURCE** | | **TRANSFORMATIONS** | **TARGET** | |
| **HR.EMPLOYEES** | |  | **HR.EMP\_DEPT\_TRG** | |
| **Column Names** | **Data Types** | **Filter Transformation** | **Column Names** | **Data Types** |
| EMPLOYEE\_ID | NUMBER(6) | Direct move | EMPLOYEE\_ID | NUMBER(6) |
| FIRST\_NAME | VARCHAR2(20) | Direct move | FIRST\_NAME | VARCHAR2(20) |
| LAST\_NAME | VARCHAR2(20) | Direct move | LAST\_NAME | VARCHAR2(20) |
| EMAIL | VARCHAR2(25) | Direct move | EMAIL | VARCHAR2(25) |
| PHONE\_NUMBER | VARCHAR2(20) | Direct move | PHONE\_NUMBER | VARCHAR2(20) |
| HIRE\_DATE | DATE | Direct move | HIRE\_DATE | DATE |
| JOB\_ID | VARCHAR2(10) | Direct move | JOB\_ID | VARCHAR2(10) |
| SALARY | NUMBER(8,2) | Direct move | SALARY | NUMBER(8,2) |
| COMMISSION\_PCT | NUMBER(2,2) | Direct move | COMMISSION\_PCT | NUMBER(2,2) |
| MANAGER\_ID | NUMBER(6) | Direct move | MANAGER\_ID | NUMBER(6) |
| DEPARTMENT\_ID | NUMBER(4) | ***Only records which have department\_id =10 will be loadeded to target,rest other records will be dropped.*** | DEPARTMENT\_ID | NUMBER(4) |
|  |  |  |  |  |
|  |  |  |  |  |
| **MAPPING-4** |  |  |  |  |
|  |  |  |  |  |
| **SOURCE** | | **TRANSFORMATIONS** | **TARGET** | |
| **HR.EMPlOYEES, HR.DEPARTMENTS** | |  | **HR.EMP\_DEPTNAME\_TRG** | |
| **Column Names** | **Data Types** | **Joiner Transformation ( EQUI JOIN)** | **Column Names** | **Data Types** |
| EMPLOYEE\_ID | NUMBER(6) | Direct move | EMPLOYEE\_ID | NUMBER(6) |
| FIRST\_NAME | VARCHAR2(20) | Direct move | FIRST\_NAME | VARCHAR2(20) |
| LAST\_NAME | VARCHAR2(20) | Direct move | LAST\_NAME | VARCHAR2(20) |
| EMAIL | VARCHAR2(25) | Direct move | EMAIL | VARCHAR2(25) |
| PHONE\_NUMBER | VARCHAR2(20) | Direct move | PHONE\_NUMBER | VARCHAR2(20) |
| HIRE\_DATE | DATE | Direct move | HIRE\_DATE | DATE |
| JOB\_ID | VARCHAR2(10) | Direct move | JOB\_ID | VARCHAR2(10) |
| SALARY | NUMBER(8,2) | Direct move | SALARY | NUMBER(8,2) |
| COMMISSION\_PCT | NUMBER(2,2) | Direct move | COMMISSION\_PCT | NUMBER(2,2) |
| MANAGER\_ID | NUMBER(6) | Direct move | MANAGER\_ID | NUMBER(6) |
| DEPARTMENT\_ID | NUMBER(4) | ***Load all the employees if Employees.Department\_id maches with Departmenets.Department\_id with department\_name*** | DEPARTMENT\_ID | NUMBER(4) |
| DEPARTMENT\_ID | NUMBER(4) | DEPARTMENT\_NAME | VARCHAR2(30) |
| DEPARTMENT\_NAME | VARCHAR2(30) |  |  |
| MANAGER\_ID | NUMBER(6) | Don’t move |  |  |
| LOCATION\_ID | NUMBER(4) | Don’t move |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| **MAPPING-5** |  |  |  |  |
|  |  |  |  |  |
| **SOURCE** | | **TRANSFORMATIONS** | **TARGET** | |
| **HR.EMPLOYEES** | |  | **HR.EMP\_COUNT\_SAL\_DEPWISE\_TRG** | |
| **Column Names** | **Data Types** | **Aggregator Transformation** | **Column Names** | **Data Types** |
| EMPLOYEE\_ID | NUMBER(6) | Don't move |  |  |
| FIRST\_NAME | VARCHAR2(20) | Don't move |  |  |
| LAST\_NAME | VARCHAR2(20) | Don't move |  |  |
| EMAIL | VARCHAR2(25) | Don't move |  |  |
| PHONE\_NUMBER | VARCHAR2(20) | Don't move |  |  |
| HIRE\_DATE | DATE | Don't move |  |  |
| JOB\_ID | VARCHAR2(10) | Don't move | NUM\_OF\_EMPLOYEES | NUMBER(5) |
| SALARY | NUMBER(8,2) | ***max(sal),min(sal), avg(sal)*** | MAXSAL | NUMBER(5,5) |
| COMMISSION\_PCT | NUMBER(2,2) | Don't move | MINSAL | NUMBER(5,5) |
| MANAGER\_ID | NUMBER(6) | Don't move | AVGSAL | NUMBER(5,5) |
| DEPARTMENT\_ID | NUMBER(4) | ***Count No.of employees department wise move them to target.*** | DEPARTMENT\_ID | NUMBER(4) |
|  |  |  |  |  |
|  |  |  |  |  |
| **MAPPING-6** |  |  |  |  |
|  |  |  |  |  |
| **SOURCE** | | **TRANSFORMATIONS** | **TARGET** | |
| **HR.EMPLOYEES** | |  | **HR.EMP\_MANAGER\_TRG** | |
| **Column Names** | **Data Types** | **Joiner Transformation** | **Column Names** | **Data Types** |
| EMPLOYEE\_ID | NUMBER(6) | Direct move | EMPLOYEE\_ID | NUMBER(6) |
| FIRST\_NAME | VARCHAR2(20) | Direct move | FIRST\_NAME | VARCHAR2(20) |
| LAST\_NAME | VARCHAR2(20) | Don't move |  |  |
| EMAIL | VARCHAR2(25) | Don't move |  |  |
| PHONE\_NUMBER | VARCHAR2(20) | Don't move |  |  |
| HIRE\_DATE | DATE | Don't move |  |  |
| JOB\_ID | VARCHAR2(10) | Don't move |  |  |
| SALARY | NUMBER(8,2) | Don't move |  |  |
| COMMISSION\_PCT | NUMBER(2,2) | Don't move |  |  |
| MANAGER\_ID | NUMBER(6) | ***Load only EMPLOYEE\_ID,FIRST\_NAME details with their MANAGER Names*** | MANAGER\_NAME | VARCHAR2(20) |
| DEPARTMENT\_ID | NUMBER(4) | Don't move |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| **MAPPING-7** |  |  |  |  |
|  |  |  |  |  |
| **SOURCE** | | **TRANSFORMATIONS** | **TARGET** | |
| **HR.EMPLOYEES** | |  | **HR.EMP\_SAL\_RANKS\_TRG** | |
| **Column Names** | **Data Types** | **Rank Transformation** | **Column Names** | **Data Types** |
| EMPLOYEE\_ID | NUMBER(6) | Direct move | EMPLOYEE\_ID | NUMBER(6) |
| FIRST\_NAME | VARCHAR2(20) | Direct move | FIRST\_NAME | VARCHAR2(20) |
| LAST\_NAME | VARCHAR2(20) | Don't move |  |  |
| EMAIL | VARCHAR2(25) | Don't move |  |  |
| PHONE\_NUMBER | VARCHAR2(20) | Don't move |  |  |
| HIRE\_DATE | DATE | Don't move |  |  |
| JOB\_ID | VARCHAR2(10) | Don't move |  |  |
| SALARY | NUMBER(8,2) | Direct move | SALARY | NUMBER(8,2) |
| COMMISSION\_PCT | NUMBER(2,2) | ***rank() over( order by salary)*** | RANK | NUMBER(2) |
| MANAGER\_ID | NUMBER(6) | ***dense\_rank() over( order by salary)*** | DENSE\_RANK | NUMBER(2) |
| DEPARTMENT\_ID | NUMBER(4) | ***row\_number() over( order by salary)*** | ROWNUMBER | NUMBER(2) |

DataValidation Quieres:

|  |  |
| --- | --- |
| **MAP1** | **SELECT EMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER , ROUND((SYSDATE-HIRE\_DATE)/365,0) EXPEREINCE,JOB\_ID,  NVL(SALARY,0) + NVL(COMMISSION\_PCT,0) TOTAL\_SALARY,  MANAGER\_ID,DEPARTMENT\_ID FROM EMPLOYEES  MINUS  SELECT EMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER ,EXPEREINCE,JOB\_ID,TOTAL\_SALARY,MANAGER\_ID,DEPARTMENT\_ID FROM EMP\_TOTSAL\_TRG  UNION  SELECT EMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER ,EXPEREINCE,JOB\_ID,TOTAL\_SALARY,MANAGER\_ID,DEPARTMENT\_ID FROM EMP\_TOTSAL\_TRG  MINUS  SELECT EMPLOYEE\_ID, FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER , ROUND((SYSDATE-HIRE\_DATE)/365,0) EXPEREINCE,JOB\_ID,  NVL(SALARY,0) + NVL(COMMISSION\_PCT,0) TOTAL\_SALARY,  MANAGER\_ID,DEPARTMENT\_ID FROM EMPLOYEES;** |
| **MAP2** | **SELECT DEPARTMENT\_ID,SUM(SALARY) SUM\_SALARY FROM EMPLOYEES GROUP BY DEPARTMENT\_ID MINUS SELECT DEPARTMENT\_ID, TOTAL\_SALARY FROM SUM\_SAL\_DEPWISE\_TRG   UNION  SELECT DEPARTMENT\_ID, TOTAL\_SALARY FROM SUM\_SAL\_DEPWISE\_TRG  MINUS SELECT DEPARTMENT\_ID,SUM(SALARY) SUM\_SALARY FROM EMPLOYEES GROUP BY DEPARTMENT\_ID;** |
| **MAP3** | **SELECT \* FROM EMPLOYEES WHERE DEPARTMENT\_ID=10 MINUS SELECT \* FROM EMP\_DEPT\_TRG  UNION  SELECT \* FROM EMP\_DEPT\_TRG MINUS SELECT \* FROM EMPLOYEES WHERE DEPARTMENT\_ID=10;** |
| **MAP4** | **SELECT E.\*,D.DEPARTMENT\_NAME FROM EMPLOYEES E, DEPARTMENTS D WHERE E.DEPARTMENT\_ID=D.DEPARTMENT\_ID MINUS SELECT \* FROM EMP\_DEPTNAME\_TRG  UNION  SELECT \* FROM EMP\_DEPTNAME\_TRG MINUS  SELECT E.\*,D.DEPARTMENT\_NAME FROM EMPLOYEES E, DEPARTMENTS D WHERE E.DEPARTMENT\_ID=D.DEPARTMENT\_ID;** |
| **MAP5** | **SELECT DEPARTMENT\_ID,COUNT(\*)NUM\_OF\_EMPLOYEES,MAX(SALARY)MAXSAL,MIN(SALARY) MINSAL,AVG(SALARY)AVGSAL FROM EMPLOYEES GROUP BY DEPARTMENT\_ID; MINUS SELECT DEPARTMENT\_ID,NUM\_OF\_EMPLOYEES,MAXSAL,MINSAL,AVGSAL FROM EMP\_COUNT\_SAL\_DEPWISE\_TRG  UNION  SELECT DEPARTMENT\_ID,NUM\_OF\_EMPLOYEES,MAXSAL,MINSAL,AVGSAL FROM EMP\_COUNT\_SAL\_DEPWISE\_TRG MINUS SELECT DEPARTMENT\_ID,COUNT(\*)NUM\_OF\_EMPLOYEES,MAX(SALARY)MAXSAL,MIN(SALARY) MINSAL,AVG(SALARY)AVGSAL FROM EMPLOYEES GROUP BY DEPARTMENT\_ID;** |
| **MAP6** | **SELECT E.EMPLOYEE\_ID,E.FIRST\_NAME,M.FIRST\_NAME MANAGER\_NAME FROM EMPLOYEES E, EMPLOYEES M WHERE E.EMPLOYEE\_ID=M.MANAGER\_ID MINUS SELECT EMPLOYEE\_ID,FIRST\_NAME,MANAGER\_NAME FROM EMP\_MANAGER\_TRG  UNION   SELECT EMPLOYEE\_ID,FIRST\_NAME,MANAGER\_NAME FROM EMP\_MANAGER\_TRG MINUS SELECT E.EMPLOYEE\_ID,E.FIRST\_NAME,M.FIRST\_NAME MANAGER\_NAME FROM EMPLOYEES E, EMPLOYEES M WHERE E.EMPLOYEE\_ID=M.MANAGER\_ID;** |
| **MAP7** | **select employee\_id, first\_name,salary,rank() over( order by salary)rank, dense\_rank() over( order by salary) dense\_rank, row\_number() over( order by salary) rownumber from employees MINUS Select employee\_id, first\_name ,salary,rank,dense\_rank,rownumber from EMP\_SAL\_RANKS\_TRG  UNION  Select employee\_id, first\_name ,salary,rank,dense\_rank,rownumber from EMP\_SAL\_RANKS\_TRG MINUS select employee\_id, first\_name,salary,rank() over( order by salary)rank, dense\_rank() over( order by salary) dense\_rank, row\_number() over( order by salary) rownumber from employees** |

Difference between rank(),

dense\_rank(),row\_number()

• select first\_name, salary, rank() over( order by salary)rank from

employees;

• select first\_name, salary, dense\_rank() over( order by salary)rank

from employees;

• select first\_name, salary, row\_number() over( order by salary)rank

from employees;

• select first\_name,salary,rank() over( order by salary)rank,

dense\_rank() over( order by salary) dense\_rank, row\_number() over(

order by salary) rownumber from employees;