Agilos®

Data warehouse

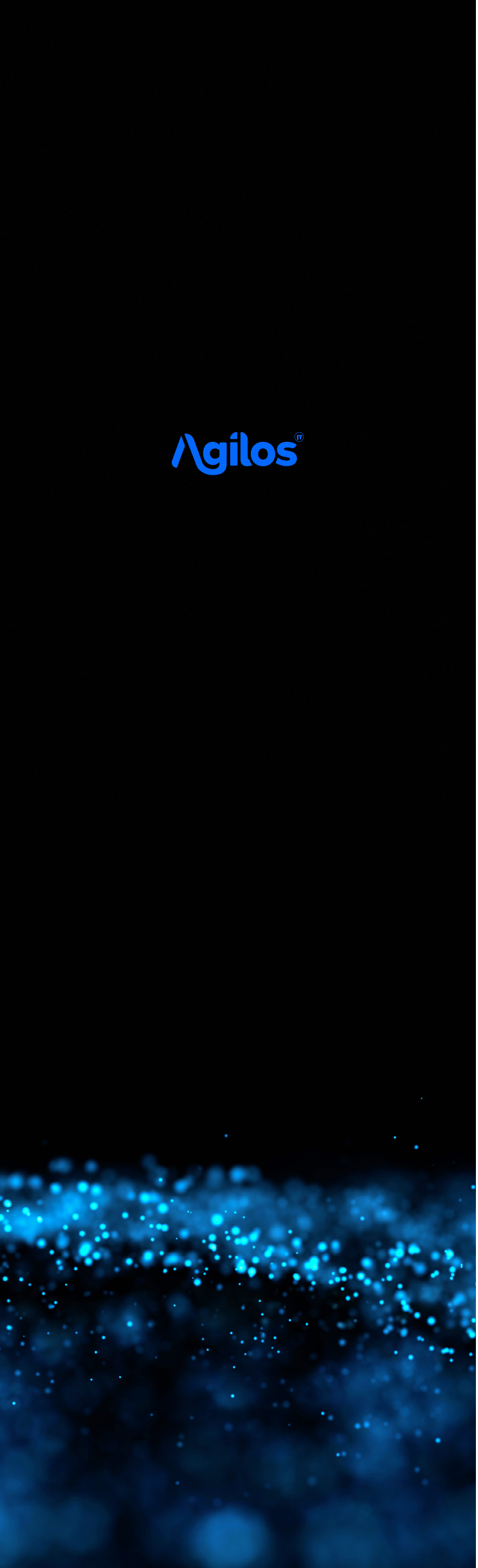
— concepts and best practices —

TABLE OF CONTENTS

Contents

TABLE OF CONTENTS

TABLE OF CONTENTS



1. Introduction

We've entered an era where data has become the most critical asset of every organization. Data-driven decision-making is at the very core of digital transformation initiatives. Organizations are increasingly relying on data to make decisions to support business objectives, such as revenue growth, profitability and customer satisfaction.

In order to provide the business with data it needs to perform decision making, it's necessary to have a well-structured data warehouse (DWH) containing high quality and reliable data. A well-crafted data governance strategy is necessary to maximize the data's value, manage its risks, and reduce the cost of its management.

For every organization, it's important to have a well-defined change and release management procedures to help deploy new changes without any disruption or downtime. Writing technical documentation and defining naming conventions is also necessary because it simplifies product maintainability and further development.

1.1. Purpose

The purpose of this document is to describe the concepts and best practices of data warehousing we've learned over many years of experience working with DWH.

Early on, we've set ourselves on a mission of defining standards and best practices of data warehousing and educating the community about it. Since our inception as a small and agile company, we've been successful in comprehending the requirements of our clients and adapting to them. This has facilitated the growth of our company and enabled us to retain a loyal client base.

This document is intended for everyone working in data warehousing and those who wish to improve their understanding of it. It's available as a free download on our web page and also as an online version. Since our company's policy is honesty and transparency, we've made this document easily accessible to everyone interested, from our colleagues in other companies to our clients. As a company, we aim to contribute as much as possible to education and knowledge transfer in order to give back to the community and encourage the development of junior consultants who are just starting out in this field.

This document, in our opinion, demonstrates our quality and expertise. We try to turn our expertise into innovation by developing tools that make it easier to work with complex business systems which use large amounts of data. By doing this, we hope to contribute to a future in which automation will free up experts for further innovation activities in the area of data management.

1.2. Scope

The document is structured into the following chapters:

#	Chapter	Description
1	Introduction	Introduction to the document – objectives, target audience, document structure.
2	Data warehouse	This chapter contains an overview of the DWH architecture, gives out definitions of key components and describes different approaches to modeling data warehouses. It also explains what an ETL process is.
3	ETL tools	This chapter gives out basic instructions for using ETL tools on an example of Oracle Data Integrator.
4	Change management	In this chapter we describe the importance and main objectives of change management. We also give out a basic overview and best practices of CM.
5	Release management	This chapter contains an overview of release management activities and describes an optimal RM model. A well-implemented RM model in an organization ensures the delivery of quality releases that are scalable for the future and leads to greater efficiency and stability.
6	SQL and procedural languages	This chapter contains a collection of tips and tricks for working with SQL and PL/SQL - best practices for writing SQL queries, using execution plans, debugging PL/SQL programs... We also define naming conventions for PL/SQL objects.
7	Technical documentation	In this chapter we describe the benefits of writing technical documentation and share some guidelines for writing excellent and useful documentation. We also share a template for writing documentation for DWH.
8	Data governance	This chapter describes the importance and different aspects of data governance. A well-crafted data governance strategy is necessary to maximize the data's value, manage its risks, and reduce the cost of its management.
9	Data pipelines	This chapter explains the concept of data pipelines and other methods of dealing with vast amounts of data besides DWH.
10	Our tools	In this chapter we give an overview of our custom tools developed for automating certain processes in dealing with data.

> Table 1-1 Document scope

1.3. Related documents

The following documents are related to this document:

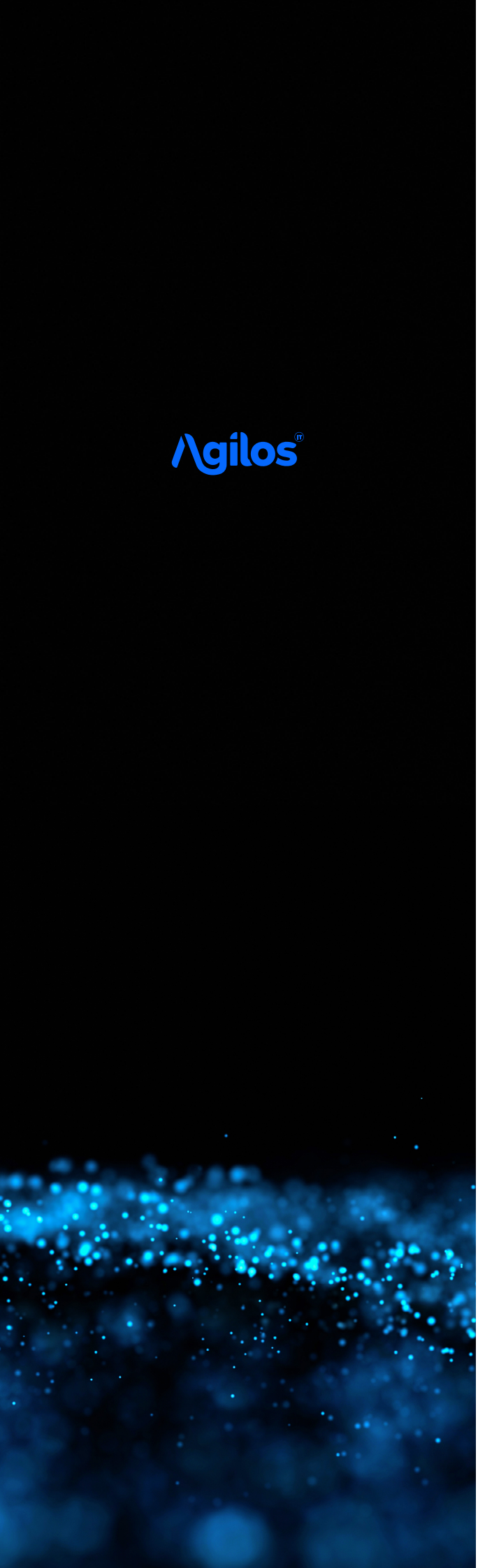
#	Document	Description	Language	Link
1	SQLtoODI whitepaper	A document which contains brief information about our tool SQLtoODI – what it's for, how it's used, main advantages, cost-benefit analysis...	Croatian	SQLtoODI whitepaper
2	DataLineage whitepaper	A document which contains brief information about our tool DataLineage – what it's for, how it's used, main advantages, cost-benefit analysis...	Croatian	DataLineage whitepaper
3	DataQuality whitepaper	A document which contains brief information about our tool DataQuality – what it's for, how it's used, main advantages, cost-benefit analysis...	Croatian	DataQuality whitepaper
4	Technical documentation template	A template for writing technical documentation for data warehouses.	English	Technical documentation template

> Table 1-2 Related documents

1.4. Acronyms

Acronym	Description
BI	Business Intelligence
CAB	Change Advisory Board
CCPA	California Consumer Privacy Act
CI	Configuration Item
CM	Change management
CMDB	Configuration Management Database
CRM	Customer Relationship Management
DB	Database
DBMS	Database Management System
DEV	Development environment
DQ	Data Quality
DWH	Data warehouse
ELT	Extract – load - transform
ERP	Enterprise Resource Planning
ETL	Extract – transform - load
GDPR	General Data Protection Regulation
IKM	Integration Knowledge Module
ITIL	Information Technology Infrastructure Library
KM	Knowledge Module
KPI	Key Performance Indicator
LKM	Loading Knowledge Module
NF	Normal Form
ODI	Oracle Data Integrator
OLAP	Online Analytical Processing
OLTP	Online Transactional Processing
OR	Operational Readiness
ORT	Operational Readiness Testing
PL/SQL	Procedural Language for SQL
PROD	Production environment
RM	Release management
QA	Quality Assurance
RFC	Request for Change document
SCD	Slowly Changing Dimension
SQL	Structured Query Language
TCPA	Telephone Consumer Protection Act
UAT	User Acceptance Testing

> Table 1-3 Acronyms



2. Data warehouse

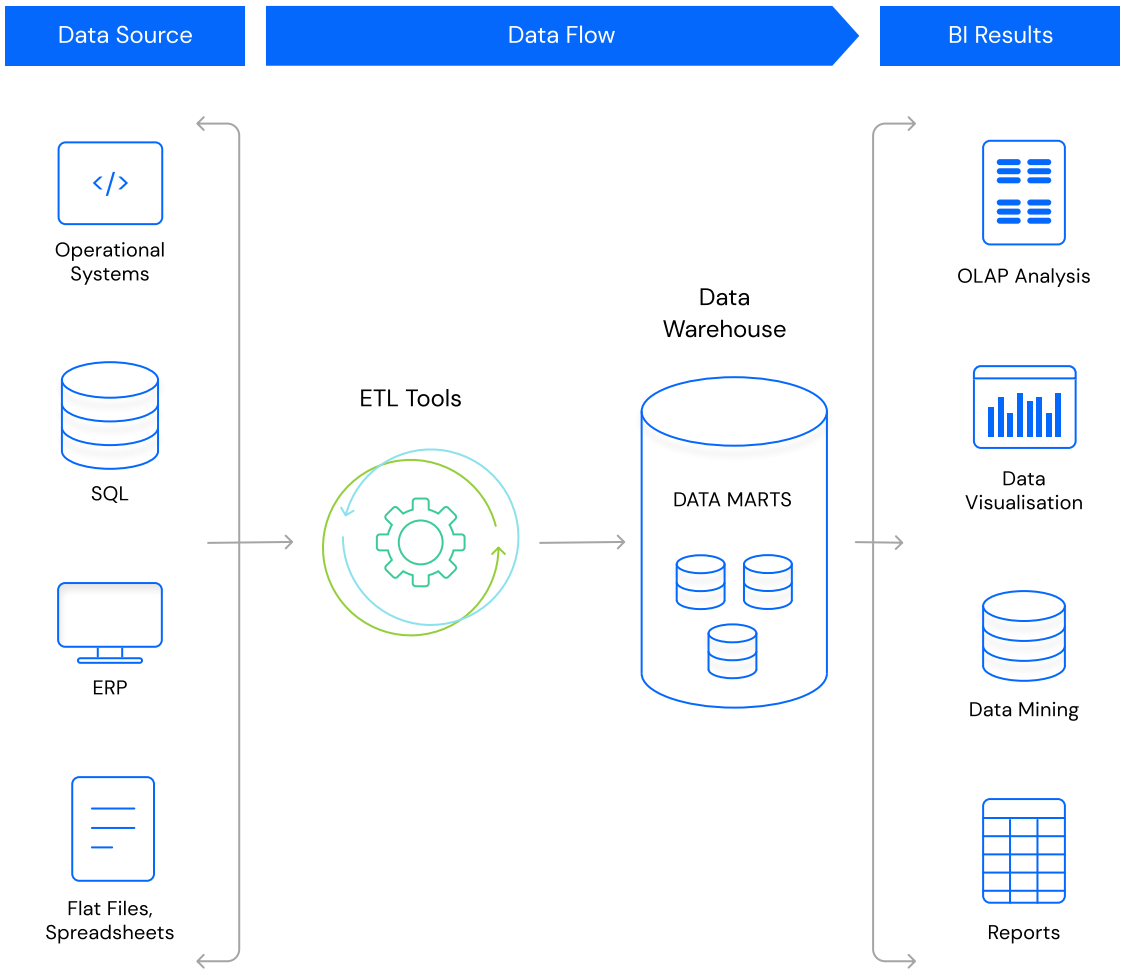
2.1. Overview

Data warehouse (DW or DWH) is a system used for reporting and data analysis and is considered a core component of business intelligence. Data warehouse centralizes and consolidates large amounts of data from multiple sources. DWH allows organizations to derive valuable business insights from their data to improve decision-making. Over time, it builds a historical record that can be invaluable to data scientists and business analysts.

A typical data warehouse often includes the following elements:

- A relational database to store and manage data
- An extraction, loading, and transformation (ELT) solution for preparing the data for analysis
- Statistical analysis, reporting, and data mining capabilities
- Client analysis tools for visualizing and presenting data to business users

2.2. DWH architecture



> Figure 2-1 Simplified DWH architecture

- **Data sources** - These are all of the systems that capture and hold the transactional and operational data identified as essential for analysis — for example, ERP, CRM, finance, manufacturing and supply chain management systems, flat files, etc. Data can be structured, semi structured or unstructured. Data sources can also include secondary sources, such as market data and customer databases from outside information providers. As a result, both internal and external data sources are often incorporated into DWH architecture.

- **Stage Area** - The source data must go through a cleansing process to eliminate any inconsistencies and fill in any missing information. The data is then integrated to bring together different sources into a uniform schema. This can be achieved through the use of ETL tools, which can gather data from various schemas and carry out tasks such as extraction, transformation, cleaning, validation, filtering, and loading of source data into a data warehouse.

- **Data Warehouse** - The data is consolidated into a single, centralized repository known as a data warehouse. This warehouse can be accessed directly or serve as a source for creating data marts, which are tailored for specific departments within the organization. To assist in managing the information, meta-data repositories store details on sources, access methods, data staging, users, data mart schemas, and more. Each data mart contains information specific to a particular function within an organization, and there can be multiple data marts depending on the number of functions within the organization. In other words, a data mart contains a portion of the data stored in the data warehouse.

- In the **presentation layer**, integrated data is efficiently and flexibly accessed to issue reports, dynamically analyze information, and simulate hypothetical business scenarios.

2.2.1. Kimball vs. Inmon

Two pioneers of data warehousing, named Bill Inmon and Ralph Kimball, had different approaches to data warehouse design.

The Inmon approach, also known as the “top-down” approach, emphasizes the use of a data warehouse as the centralized repository for all enterprise data. This approach focuses on capturing all data from various sources and storing it in a single location, in order to make it available for reporting and analysis. Dimensional data marts are then created based on the data warehouse model.

On the other hand, the Kimball approach, also known as the “bottom-up” approach, emphasizes the use of decentralized data marts. This methodology gives importance to creating smaller, specialized data marts which are designed to meet the unique requirements of different business units or departments. The data marts are then linked to the central data warehouse, allowing for a more flexible and decentralized approach to data management.

It is difficult to say which one of these two methodologies is more widely used, as it likely varies depending on the specific industry or organization. However, the Kimball methodology is often considered to be more popular among practitioners, due to its focus on creating small, subject-specific data marts that are tailored to the specific needs of different business units or departments. This approach is seen as more flexible and better suited to the changing needs of the business, as opposed to the Inmon approach, which emphasizes the centralization of data in a “corporate data warehouse”. That being said, both methodologies have their own advantages and disadvantages and organizations may use a hybrid approach which combines elements from both methodologies. Ultimately, the choice of which one to use will depend on the specific needs and constraints of the organization.

2.3. Dimensional modeling

Dimensional modeling is a data structure technique optimized for data storage in a data warehouse. The purpose of dimensional modeling is to optimize the database for faster retrieval of data. The concept of dimensional modeling was developed by Ralph Kimball and consists of "**fact**" and "**dimension**" tables.

- **Fact** table represents the measurements, metrics or facts of a business process. It usually contains numerical data which needs to be analyzed. For example, a number of products sold in a specific time period.
- **Dimension** tables are business descriptions which represent context of facts, for example, product name.

2.3.1. Dimensional modeling process

The dimensional data model is constructed using a star schema, with a fact table at the center and a number of dimension tables surrounding it. Dimensional modeling design usually follows a four-step procedure:

1. **Select the business process to model** - the first step is to decide what business process to model by gathering and comprehending business needs and available data (examples of business processes are order processing, shipments, materials purchasing, etc.)

2. **Declare the grain** - precisely describe what each record in a fact table represents. The level of detail associated with the facts in the fact table is expressed by the grains.

3. **Identify the dimensions** - adding a number of dimensions that represent all possible descriptions that take on single values in the context of each fact in the fact table (examples of common dimensions are date, time, product, customer, store, etc.)

4. **Identify the fact** - select the numeric facts which will be loaded into the fact table. In order to identify the facts, we need to identify the business process's key performance indicators (KPIs) or find out what we are trying to measure.

2.3.2. Benefits of dimensional modeling

The dimensional model has the following benefits:

- it has proven to be **easier to understand** because data is organized into coherent dimensions, making it easier for business users to analyze the data
- it **improves query performance**:
- the dimensional model is more denormalized therefore it is optimized for querying
- the dimensional model's predictable framework enables the database engine to make substantial assumptions about the data
- it is **easily extensible**

2.3.3. Dimension tables

Columns in a dimension table represent dimensions that provide the necessary context for studying the facts. Attributes that describe facts are typically stored in a dimension table. A dimension table usually has numerous columns, each dedicated to a specific attribute.

The dimension table is not necessary in the third normal form (3NF). The primary key of a dimension table is a single surrogate key that is a part of the composite primary key of the fact table.

2.3.3.1. Surrogate keys in dimension tables

The value of the primary key in a dimension table must remain unchanged. Also, it is highly recommended that all dimension tables use surrogate keys as primary keys.

Surrogate keys are keys generated and managed inside the data warehouse rather than keys extracted from data source systems.

There are several advantages of using surrogate keys in dimension tables:

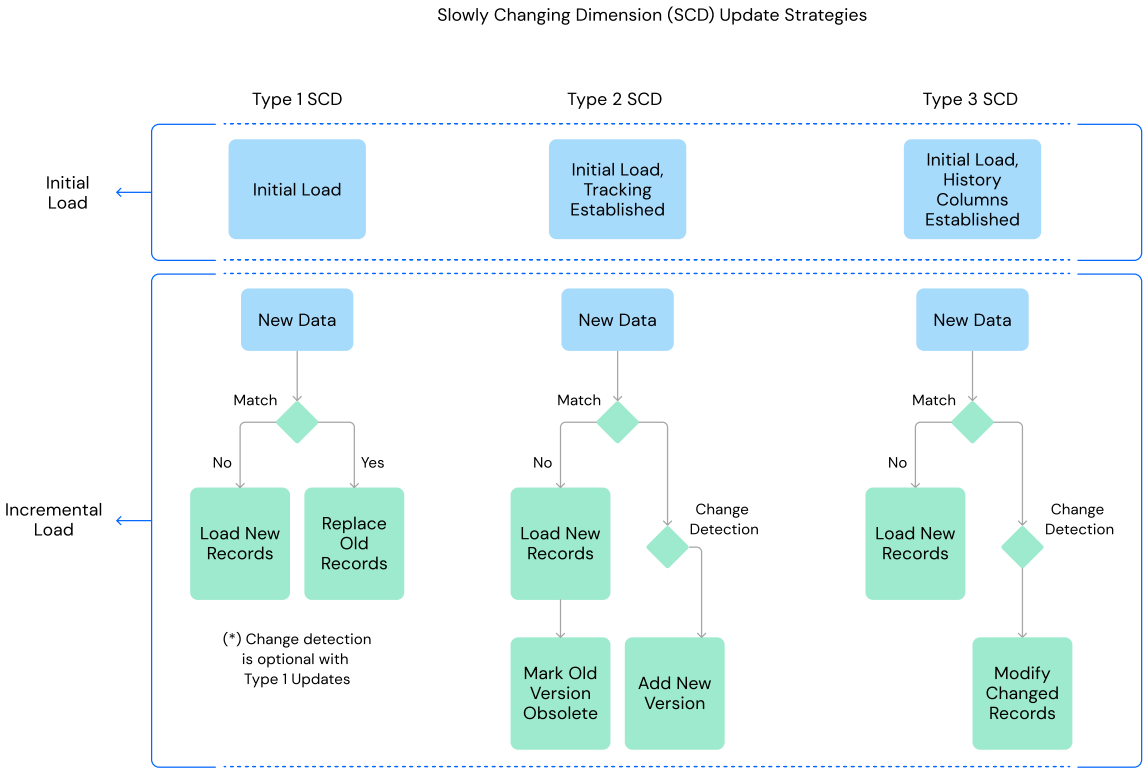
- **Performance** – join processing between dimension tables and fact table is much more efficient by using a single field surrogate key.
- **Integration** – in terms of data acquisition, the surrogate key makes it possible to integrate data from multiple data sources even if they lack consistent source keys.

We should manage data versions – by keeping track of changes in dimension field values inside the dimension table.

Dimension tables must be designed in a way that they can be easily shared between multiple data marts and cubes within a data warehouse. This guarantees that the data warehouse provides consistent information for similar queries. Additionally, surrogate keys need to be used as the primary keys of dimension tables in order to enable easier sharing of the dimension tables.

2.3.3.2. Slowly changing dimension

The attributes of a given record in the dimension table could be changed (for example: product description, shipping address). This is known as **slowly changing dimension** and there are methods for effectively addressing each type of slowly changing dimension.



> Figure 2-2 SCD overview

Types of slowly changing dimensions:

- **Type 1 (SCD1)** is used when the history of the data is irrelevant. The corresponding dimension attribute is overwritten whenever the data in the data source is changed.

Product Dim (Source)		
Product Name	**Product ID**	**Product Desc**
10 inch box	010	10 inch ~~glued~~ box 10 inch **pasted** box
12 inch box	012	12 inch glued box

Product Dim (Target)		
Product Name	**SID**	**Product Desc**
10 inch box	010	10 inch **pasted** box
12 inch box	012	12 inch glued box

> Figure 2-3 SCD1 example

- **Type 2 (SCD2)** is used when the change of data in the data source is important and you want to preserve the historic context of facts corresponding to the changing data. When data in the data source changes, a new row is inserted into the dimension table. The previous row remains unchanged.

- **Type 3 (SCD3)** happens when you want to learn about every fact before and after the attribute changes. To deal with this, you can introduce a new attribute to the existing row and update the value to both fields.

Product Dim (Source)		
Product Name	**Product ID**	**Product Desc**
12inch box	012	12 inch glued box
10 inch box	010	10 inch ~~glued~~ box 10 inch **pasted** box

Product Dim (Target)					
SID	**Source Product ID**	**Product Name**	**Product Desc**	**EFF_START_DT**	**EFF_END_DT**
0001	012	12 inch box	12 inch glued box	Jan-01-1753	Dec-31-9999
0002	010	10 inch box	10 inch box	Jan-01-1753	May-12-06
0003	010	10 inch box	12 inch pasted box	Mauy-12-06	Dec-31-9999

> Figure 2-4 SCD2 example

There are a few more SCD types, mostly hybrid versions of the previous three, but they are not as commonly used as Types 1-3 so we won't explain them in detail.

Before Change

Empty ID	Name	Current_Department	Previous_Department
1234	Becham	Data Engineer	Data Engineer

After Change

Empty ID	Name	Current_Department	Previous_Department
1234	Becham	Data Engineer	Data Engineer
1234	Becham	Software Engineer	Data Engineer

> Figure 2-5 SCD3 example

2.3.4. Fact table

The fact table, which stores facts or measures of interest, is the center of the star schema. Typically, facts are numbers which can be summarized, aggregated, or rolled up.

The fact table contains surrogate keys as a part of its primary key. Those keys are the foreign key of the dimension tables, which looks at the exact version of the data in the dimension table.

2.3.4.1. Measure types

A fact table can store different types of measures such as additive, non-additive, semi-additive.

- **Additive** – as its name implies, additive measures can be summed across any of the dimensions associated with the fact table. For example: summation of sales in a year.
- **Semi-additive** – semi-additive measures can be summed across some dimensions, but not all. For example, balance amounts are semi-additive because they are additive across all dimensions except time.
- **Non-additive** – non-additive measures cannot be summed across any dimensions, for example, ratios. Whenever possible, it is a good idea to store the non-additive measure's fully additive components and add their sums to the answer set, before calculating the final non-additive fact.

2.3.4.2. Types of fact tables

All fact tables are categorized by the three most basic measurement events:

- **Transactional** – Transactional fact table is the most basic one, in which each grain is shown as "one row per line in a transaction," like how each line item appears on an invoice. The most detailed level of data is stored in a transactional fact table; consequently, it is associated with a significant number of dimensions.
- **Periodic snapshots** – Periodic snapshots fact table stores the data that is a snapshot in a period of time. Data from a transaction fact table, where you can select a time to acquire the output, is used as the source data for the periodic snapshots' fact table.
- **Accumulating snapshots** – The accumulating snapshots fact table describes the activity of a business process that has a clear beginning and end. Therefore, this kind of fact table has many date columns to reflect significant turning points in the process. Processing of a material is a nice example of accumulating snapshot fact tables. When a step in handling the material is complete, the appropriate record in the fact table for accumulating snapshots is updated.

2.3.5. Dealing with NULL values

We will describe two cases where null values should be avoided in a dimensional model. In these situations, using default values instead of nulls is recommended.

2.3.5.1. Handling Null Foreign Keys in Fact Tables

The first case where nulls should be avoided is when we encounter a null value as a foreign key for a fact table row. We have to do something in this situation because a null value in a fact table foreign key field will violate referential integrity. There are a number of reasons why this could happen, but, regardless of the reason, null values should be replaced with default values to prevent any further issues.

When dealing with null foreign keys, we recommend using as much intelligence as possible during the ETL process to select a default dimension row that has meaning for business users. Don't just create one default row and point all default scenarios to it. To provide the most comprehensive understanding of the data possible, take into consideration each condition separately and provide as many default rows as are necessary.

Some examples of default rows include Missing Value, Not Happened Yet, Bad Value, Not Applicable. Usually, the ETL team assigns specific values such as 0, -1, -2, and -3 to the keys that describe these alternatives.

2.3.5.2. Handling Null Attribute Values in Dimension Tables

We should also avoid using nulls when we can't provide a value for a dimension attribute in a valid dimension row. The value of a dimension attribute may not be accessible for a variety of reasons, including the following:

- Missing Value - The attribute was not present in the source data.
- Not Happened Yet – Due to issues with the timing of the source system, the attribute is not yet available.
- Domain Violation - We either have a problem with the quality of our data or we don't know all the business rules that apply to the attribute. The data that the source system provided is either not valid for the column type or falls outside the list of domain values that are acceptable.
- Not Applicable - The attribute does not apply to the dimension row in question.

In most cases, the actual values that describe the null conditions can be found in the text attributes of dimension tables. Try to think about how this will affect the BI tools down the road which need to display your unique null value description in a report with a fixed format. Avoid things as filling the default attributes with a space or a meaningless string of characters like @@@, as these could only confuse business users. Provide as much meaning as possible for each dimension attribute's default values to provide context to business users.

2.3.6. Star schema

Star schema is a dimensional design system for a relational database, often used in data warehouse systems. A fact table is located at the core of the star schema, and several dimension tables surround it. The name "star schema" refers to the structure's star-like appearance.

In the star schema, related dimensions are grouped as columns in dimension tables and used to store the context of the facts stored in the fact table.

2.3.6.1. Star schema example

The following is an example of a star schema based on dimensions and facts for the quarterly balance of funds across products and customer segments. A periodic snapshot of the account balance for accounts belonging to various products and customers will be uploaded at intervals of quarters.

> Figure 2-6 Star schema example

Let's take a look at the star schema example above in greater detail:

- **Fact table** called F_BALANCE is at the center of the schema. The primary key of the fact table contains four surrogate keys associated with dimension tables: *CustomerKey*, *DateKey*, *AccountKey* and *ProductKey*. The fields *Currency* and *Balance* are used to store facts.
- Surrounding the fact table are **dimension tables:** D_CUSTOMER, D_DATE, D_PRODUCT and D_ACCOUNT.

By examining several dimensions, star schema can assist business analysts in providing answers to queries that might not have been raised during the design phase.

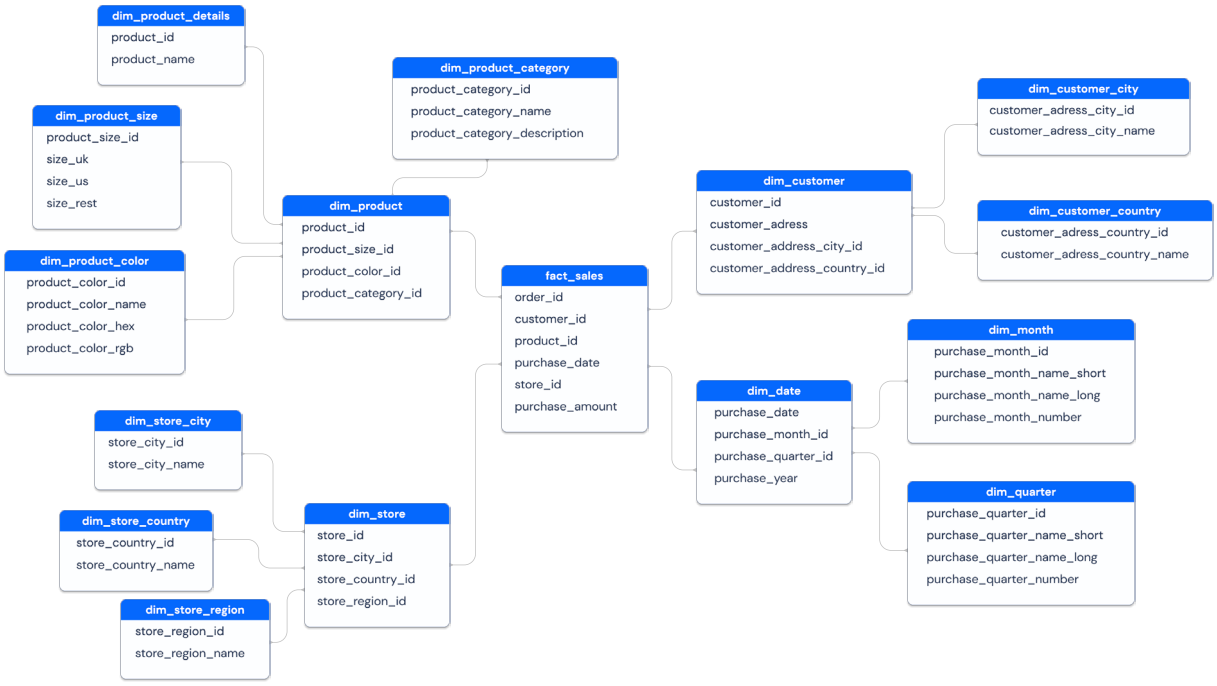
Star schema frequently stores data at a high level of detail, while aggregations allow data to be rolled up at different levels of detail. The ability to study facts depends on the level of detail that the fact table stores.

The star schema offers additional reporting possibilities the more dimension tables it contains.

2.3.7. Snowflake schema

The snowflake schema is a variant of the star schema model where the dimension tables are normalized by further dividing the records into additional tables. Star schemas' dimension tables can be normalized using the snowflaking technique. The resulting structure, after all the dimension tables are completely normalized, resembles a snowflake with the fact table in the middle.

The snowflake schema includes one fact table which is connected to several dimension tables, which can be connected to other dimension tables through a many-to-one relationship. Tables in a snowflake schema are usually normalized to the third normal form. Each dimension table implements exactly one level in a hierarchy.



> Figure 2-7 Snowflake schema

2.3.7.1. Snowflake schema advantages and disadvantages

Advantages:

- Due to normalization, which is the fundamental quality of a snowflake schema, there should be little or no redundancy.
- Data quality will be exceptional, as normalization grants the benefit for the well-defined form of tables/data.
- When queried with joins, clear and accurate data is retrieved.
- High data quality and accuracy helps in facilitating efficient reporting and analysis.

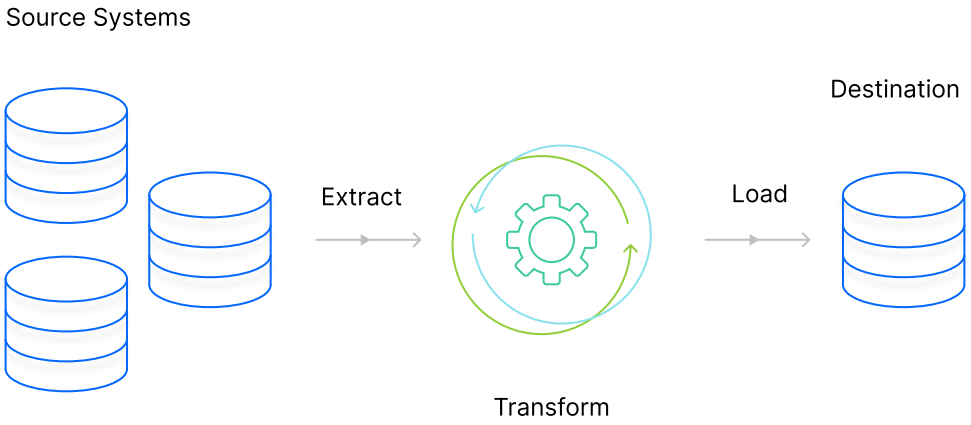
Disadvantages:

- Snowflake schema is a complex system, as it can have any number of levels of normalization depending on the depth of the given database.
- If any new business requirement creates a need for denormalization, data quality will be taken back and redundancy may occur. This may lead to restructuring the entire schema.
- Maintenance is difficult as the higher-level dimensions need to be expanded constantly.
- Low performance as it requires complex join queries.

2.4. ETL (Extract – transform – load)

ETL stands for “Extract - transform – load”. A typical ETL process collects and refines various types of data which are then sent to a data warehouse (or data lake). By allowing businesses to consolidate data from multiple data sources into a single, centralized location, ETL tools make data integration strategies possible. Additionally, ETL tools make combining different types of data possible.

The ETL process consists of three steps that enable data integration from source to destination: data extraction, data transformation, and data loading.



> Figure 2-8 ETL

2.4.1. Data extraction

During data extraction, ETL identifies the data and copies it from its sources in order to transport the data to the target datastore. The data can come from structured and unstructured sources, including documents, emails, business applications, databases, equipment, sensors, third parties, and more.

Although it can be done manually, hand-coded data extraction can be time-intensive and prone to errors. This is where ETL tools come in - they automate the extraction process and create a more efficient and reliable workflow.

2.4.2. Data transformation

The extracted data is raw in its original form; therefore, it needs to be cleansed, mapped and transformed before it can be used in the final datastore. The process of data transformation consists of several sub-processes:

- **Cleaning**— inconsistencies and missing values in the data are resolved
- **Standardization** — formatting rules are applied to the dataset
- **Deduplication** — redundant data is excluded or discarded
- **Verification** — unusable data is removed and anomalies are flagged
- **Sorting** — data is organized according to type
- **Other tasks** — any additional/optional rules can be applied to improve data quality.

Transformation is generally considered to be the most important part of the ETL process. Data transformation enhances data integrity by eliminating duplicates and ensuring that raw data arrives at its new destination fully compatible and ready to use.

2.4.3. Data loading

The newly modified data needs to be loaded into a new destination (data lake or data warehouse) as the last step in the ETL process. Data can be loaded all at once (full load) or at scheduled intervals (incremental load).

In an ETL **full loading** scenario, everything that comes from the transformation assembly line goes into new, unique records in the data warehouse or data repository. Even though there are times when this type of data loading is useful for research purposes, full loading creates datasets that increase exponentially and can quickly become challenging to maintain.

Incremental loading is a less comprehensive but more manageable approach. When data is loaded incrementally, it is compared to what already exists in the target datastore and only creates additional records when new and unique data is discovered. This type of architecture allows smaller, less expensive data warehouses to maintain and manage business intelligence.

2.4.4. ETL vs. ELT

ETL and ELT differ on two main points:

- When the transformation takes place
- Where the transformation takes place

In a traditional data warehouse, data is first extracted from source systems (ERP systems, CRM systems, etc.). Standardizing dataset dimensions is necessary for OLAP tools and SQL queries to produce aggregated results. This means that the data must undergo a series of transformations. Traditionally, these transformations have been done before the data was loaded into the target system.

However, it has become possible to implement transformations within the target system as the underlying data storage and processing technologies that support data warehousing advance. Staging areas are used in both ETL and ELT processes. In ETL, these areas are found in the tool, whether it is proprietary or custom. They are situated between the source system (for example, an ERP system) and the target system (the data warehouse).

On the other hand, with ELTs, the staging area is in the data warehouse, and opposed to using an ETL tool, the transformations are carried out by the database engine that powers the DBMS. As a result, one of the immediate consequences of ELTs is that you will no longer have access to the data preparation and cleaning functions that ETL tools provide to support the data transformation process.

2.5. Naming conventions

Some advantages of having naming conventions are:

- readability
- developers can focus on the necessary details instead of naming things
- code is easier to maintain among multiple developers.

It is important to maintain one consistent letter case for both entity and column names - we will be using all uppercase. To improve readability, underscores will be used as word separators. It is a common practice to utilize nouns or noun phrases, in their singular form, as object names.

Some systems enforce character limit on object names, e.g., Oracle 12.1 and below only allows for a maximum object name length of 30 bytes. Therefore, abbreviations and acronyms may be taken into consideration during the object naming process, despite the fact that they can often lead to misinterpretation.

In logical models, it is advisable that object names are as self-explanatory as possible, i.e., most words should be fully spelled out, except common abbreviations for longer words such as “dept” for “department” or “org” for “organization”. However, abbreviations and acronyms are typically used in physical models, to keep object names short.

Object type	Naming convention	Example
Dimension table	D_*	D_PRODUCT
Fact table	F_*	F_BALANCE
External tables (from imported csv files)	EXT_*	EXT_SALES
Temporary tables	TMP_*	TMP_STORE
Source tables	S_*	S_CURR_CODE
Auxiliary tables	AUX_*	AUX_WH
Packages	PKG_*	PKG_UTIL
Procedures	PRC_*	PRC_CLOSE_ORDERS
Functions	F_*	F_GET_NEXT
Jobs	J_*	J_CLOSE_ORDERS
Types	T_*	T_ASN_CTN
Triggers	TRG_*	TRG_FEE
Sequences	SEQ_*	SEQ_ITEM_ID
Views	VW_*	VW_CLIENT
Materialized views	MV_*	MV_ITEM_LOC
Index on partitioned column	PIX_table_name_x (x = 1..n)	PIX_HIRE_DATE
Index used specifically for ETL	EIX_table_name_x (x = 1..n)	EIX_TXN
Index created for reporting purposes	RIX_table_name_x (x = 1..n)	RIX_STATUS
Other indexes	IX_table_name_x (x = 1..n)	IX_DOC_ID

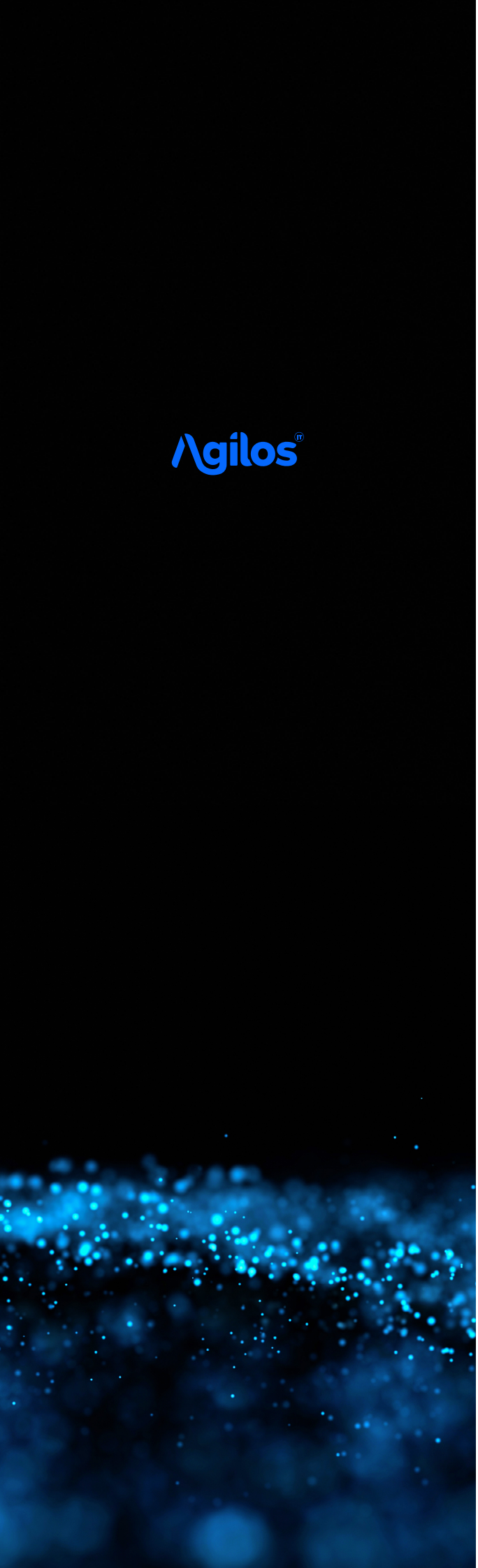
> Table 2-1 Naming conventions

Table columns:

- Column names need to be descriptive
- Column names containing the same information should be named the same in all the tables
- Column names with common features should have the same prefixes/suffixes, e.g.:
  - Time data (start_DATE, end_DATE or DATE_created, DATE_modified,...)
  - Account balance (accounting_BALANCE, loan_BALANCE, overdraft_BALANCE,...)
  - Transactional amounts (transactional_AMOUNT, overdraft_AMOUNT, installment_AMOUNT,...)
- Columns containing different levels of data aggregation shouldn't have the same name (e.g., daily, weekly, monthly average shouldn't all be called average_balance_fc but daily_avg_balance_fc, weekly_avg_balance_fc, monthly_avg_balance_fc, respectively)

Data format standardization:

- Data format should be the same for equal types of data
  - Financial data from source, for example: NUMBER (19,6)
  - Aggregated and summary data, for example: NUMBER (19,2)
- Currency columns should be VARCHAR2 (3 CHAR) because they contain ISO currency code
- Use CHAR instead of BYTE when defining string columns: 1CHAR <> 1BYTE for some special characters (like Č, Š, ć, đ, Ž, ...)
- Oracle recommends using VARCHAR2 instead of VARCHAR
- VARCHAR2 is a dynamic data type and will save the exact number of characters contained in input string unlike CHAR which saves fixed length



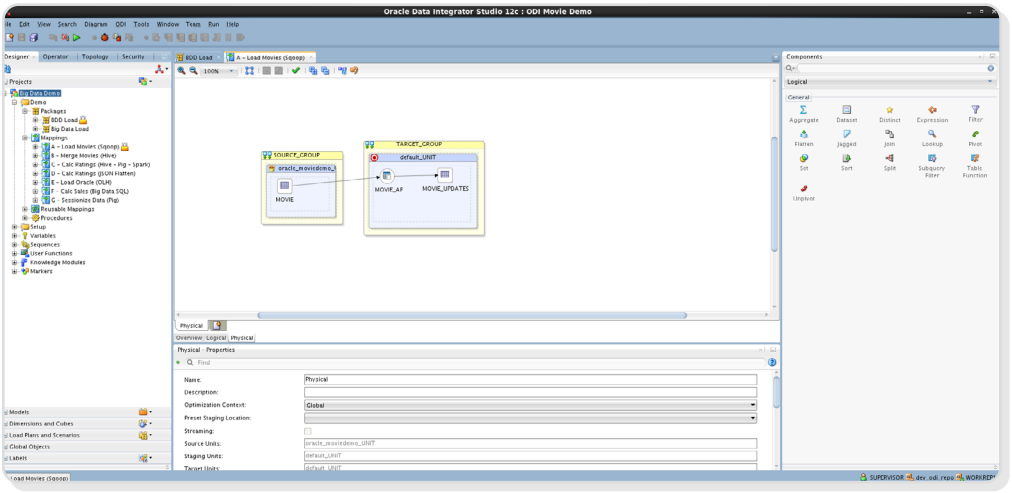
3. ETL tools

ETL tools are software solutions used to extract data from various sources, transform it into a consistent format, and load it into a target system such as a data warehouse, database, or business intelligence platform. Some popular ETL tools include: Oracle Data Integrator (ODI), Informatica PowerCenter, Microsoft SQL Server Integration Services (SSIS), IBM InfoSphere DataStage; open-source tools like Talend, Pentaho... These tools vary in terms of their features, pricing, and target audience. In this chapter, we will give an overview of one ETL tool – ODI.

When selecting an ETL tool, it's important to consider the specific needs of your organization and the compatibility of the tool with your existing systems and data sources.

3.1. Oracle Data Integrator

This is an example of a typical ODI workspace view. On the left, we can see all the projects we have in our workspace (packages, mappings, scenarios, procedures, sequences...). When we click on an object, it extends on the right part of the screen in a separate window. In this example we can see a mapping and all corresponding objects.



> Figure 3-1 Oracle Data Integrator

3.1.1. Standard ODI model organization

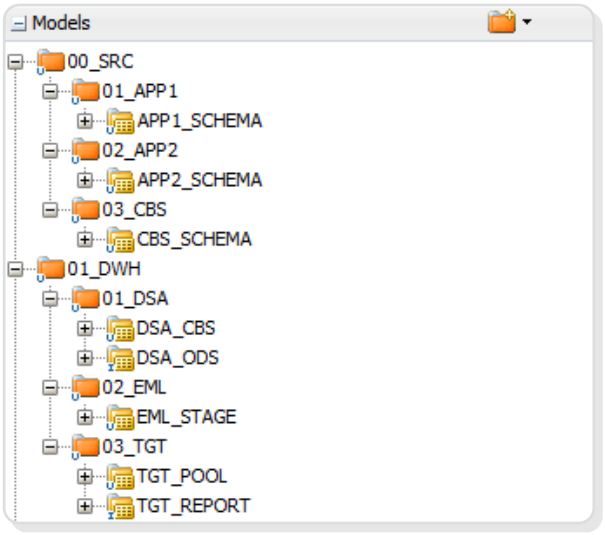
It's important to organize models well when setting up ODI development rules to ensure better management, maintenance and structure.

Models should be broken down into logical units and subunits using ODI directory. If necessary, sub-units could be broken down into more sub-units.

Models shouldn't be renamed to avoid conflicting NAME and CODE values which can cause problems with scripting. Space character shouldn't be used in model names. Best practice is to name the models after the database tables from which they are created.

Each model should have defined sub-models with tables included according to a specified mask to ensure better structure and maintenance. Sub-models should be named after the specified schema and mask.

Example of model structure inside directories prefixed with order numbers to ensure order and structure.



> Figure 3-2 ODI – model organization structure

3.1.2. Standard ODI project organization

It's important to organize projects well when setting up ODI development rules to ensure better management, maintenance and structure.

Projects should be separated into logical units like "DSA", "EML", "DataMart 1", "DataMart 2", ... "DataMart N".

Inside each project, code should be broken down into logical units and subunits using the ODI directory. If necessary, sub-units could be broken down into more sub-units.

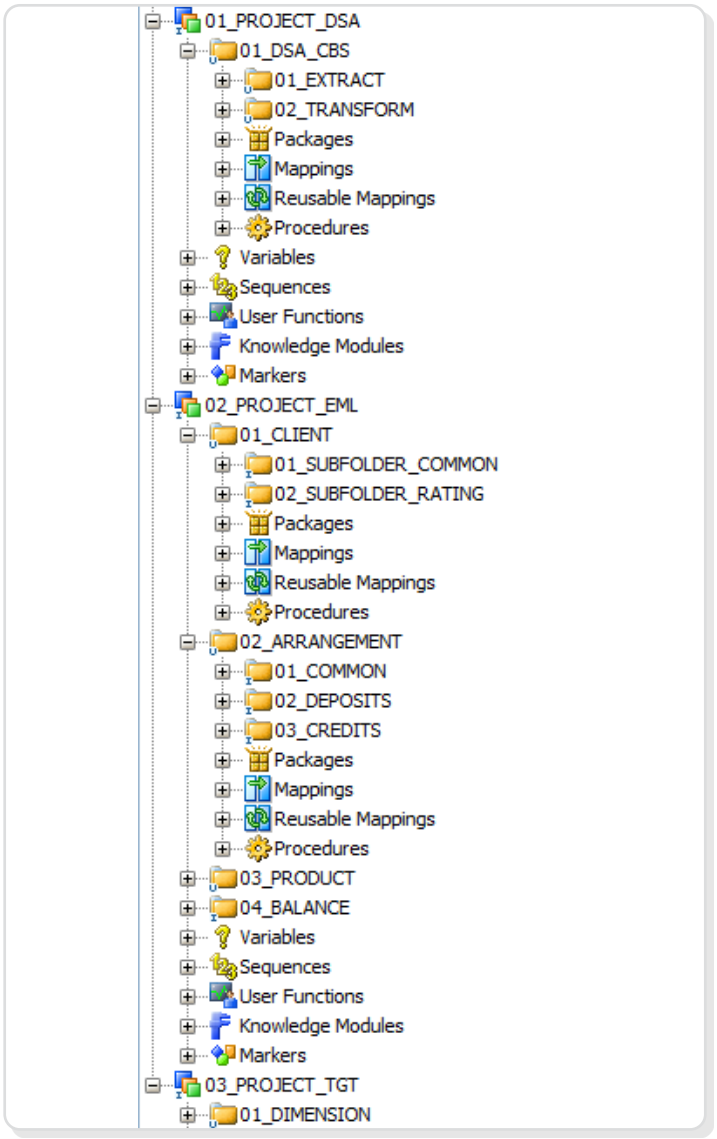
Project and directory names should be spelled with capital letters to ensure better readability. Projects shouldn't be renamed to avoid conflicting NAME and CODE values which can cause problems with scripting.

ODI objects, sequences, variables, functions and KMs should be saved inside each project only if they are unique and specific for that project. Otherwise, they should be defined as global objects.

ODI objects, packages, mappings, reusable mappings and procedures should be placed in the position where they are used according to the hierarchy of the project structure. Objects that are used in multiple directories should be placed in separate directories under the project.

In ODI you can only move objects within the same project. If moving objects from one project to another, you should use export and import functions.

Example of project structure inside directories prefixed with order numbers to ensure order and structure:



> Figure 3-3 ODI – project schema structure

3.1.3. Oracle Data Integrator development tips

1. **Knowledge Module modifications before starting ODI development**

   Loading Knowledge Modules and Integration Knowledge Modules that come with standard ODI installation should be modified before use in development. Modifications are made for:

   - optimization
     - steps or queries are optimized to run faster
     - steps that are not necessary are removed
     - parameters are added to steps using options
     - hint support is added if necessary
   - adjusting for organization specific work standards
     - for example, IKM for SCD2 flag IS_CURRENT for versioning 0/1 or Y/N

2. **Knowledge Module modifications control while working**

   Developers can use existing knowledge modules but shouldn't modify them. If there is a need for KM modification, it should first be discussed with the development team to make sure there is no other way to achieve the desired functionality. If there really is no other way to achieve the necessary functionality, then the team designated to KM modifications should make the needed adjustments.

   If the stated procedure is not followed, there is a possibility of causing errors by modifying existing KMs that are used in other parts of the project. Another possibility that can happen is the growing number of KMs gets too big to manage and slows down the development.

3. **Using capital letters for object name spelling**

   To achieve greater readability inside the repository structure all objects should be spelled with capital letters. Doing this can also help with searching through repositories. Standardizing naming logic also helps when using scripting languages. Renaming objects down the line is a complex operation that can easily cause conflicts on other objects that reference them. That's why naming standards should be defined as soon as possible.

4. **Optimization Context set-up**

   In case of conflicting context names (internal ODI code) on TEST and PROD environments you should change the Optimization Context on the mapping and then regenerate the scenario after migration.

After exporting a single or a set of mappings from one environment to another you can use a script to set appropriate Context Optimization and a second script after that to regenerate the scenarios. Using scripts speeds up and automates the whole process.

5. **Object propagation through environments**

   All new ODI objects and their scenarios should be created on the DEV environment and then propagated (export/import) on to other environments. By doing it this way you make sure that matching objects on all projects have matching internal IDs and that all possible continuity errors are avoided when propagating changes on existing objects.

6. **Using Groovy scripts**

   Groovy is a scripting language used inside ODI that can be used for any generic purpose. Using Groovy allows you to script bulk data modifications and processing. Using it should be relatively simple but writing scripts requires in-depth knowledge of ODI processes and structures. Groovy scripts should be used as much as possible because they speed up and automate data integration processes.

7. **Context Independent Design**

   Hardcoded Context should never be used in development, you should use Execution Context. This way we make sure that the object propagates through the whole code while executing it on a specific context and that the whole code executes on specified context.

8. **Using SQL scripts in ODI procedures**

   SQL scripts for ETL tasks shouldn't be used within ODI procedures. All ETL functionality should be done only with mappings to enable easier maintenance and readability.

9. **Testing mappings**

   For testing and optimization purposes, options for creating multiple physical designs should be used before choosing the optimal design. A different physical design should be set on a mapping inside of the package while making minimal adjustments during development and simplifying regeneration of the optimal solution because it's not deleted or modified.

10. **Developer side mapping validation**

    After each change is made to the logical layer; the physical layer should be carefully inspected. If the physical design is not generic, it can get invalidated or, even worse, integration logic can get messed up without the developer even realizing.

11. **Validation and mapping deployment into production**

    Two types of validation are required before deployment - physical and logical design validation. It's possible to write a groovy script for mapping validation with parameters that allow you to validate a whole folder or subfolder at once. If a certain mapping has more than one physical design and only one is supposed to go to production, all of the others should be deleted to avoid conflicts in later development.

12. **Internal code versioning**

    ODI allows internal code versioning by saving objects from work repository into master repository.

    For any object you can:

    - Create a new version
    - Inspect old versions
    - Compare object versions to older versions
    - Restore an older version of an object

    In case the PROD environment is used for versioning, you should version only parts of the code that are meant to go into production. In that case versions should be made:

    - Version of old code before importing new
    - or version of new code after the import

    In this case you should have an overview of all versions that were put into production.

    In case the DEV environment is used for versioning, you could create as many versions as you need without migrating all of them to production. In this case, you should have an overview of all the versions of an object including the ones that didn't make it into production. To be able to reconstruct older versions of production code, a careful track of versions that went to production should be kept.

13. **SQLtoODI** – if you're interested in faster development of ODI mappings, check out our SQLtoODI tool.

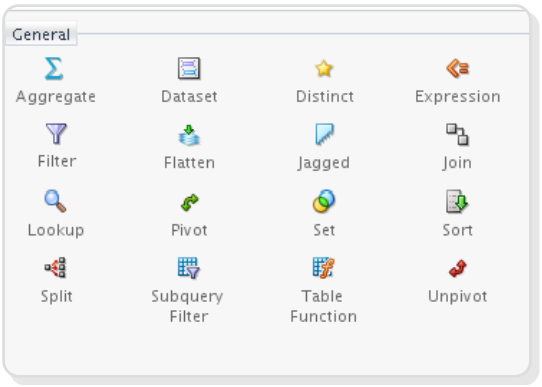
3.1.4. ODI naming conventions

3.1.4.1. Basic object naming rules

- Operators inside ODI mappings should be spelled with capital letters for better visibility.
- Object names shouldn't contain diacritics!
- Prefix G is added to GLOBAL OBJECTS for better distinction, for instance a variable should be spelled GV_%. Global objects can contain sequences, variables, knowledge modules, user functions and reusable mappings. Global objects can be used in all projects.
- When creating a function, a descriptive and meaningful name should be used without using the name of the function's schema. This enables us to move functions between folders simply by adapting the function code to the new environment.

3.1.4.2. Standard ODI mapping operator naming

Below is a picture of standard ODI operators.



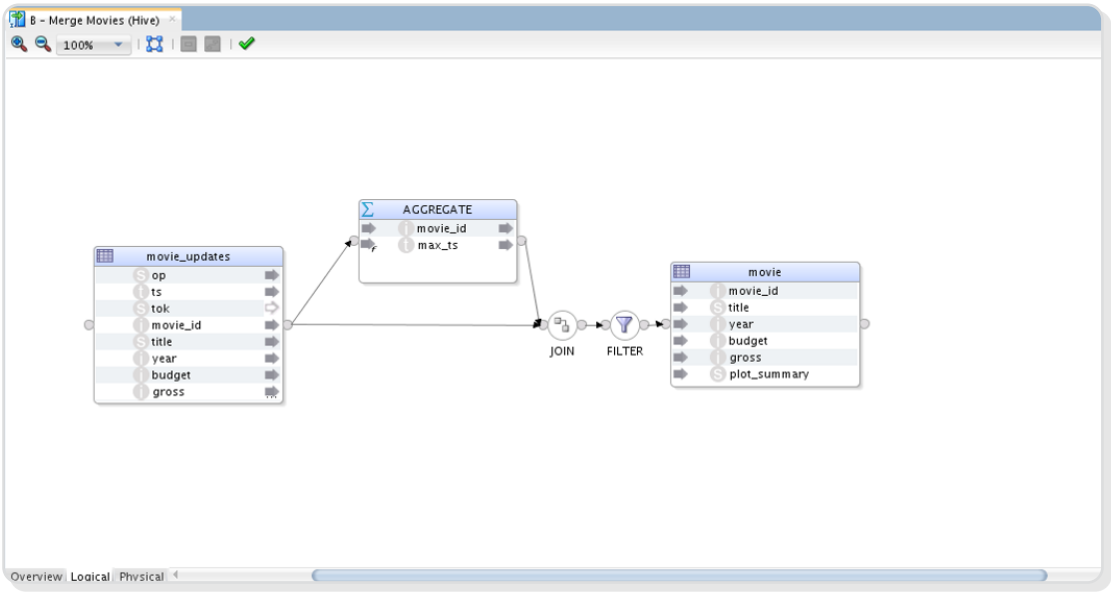
> Figure 3-4 ODI operators

Operator	Prefix	NOTE
AGREGATOR	AGG_	
DATASET	DTS_	
DISTINCT	DST_	
FILTER	FLT_	
EXPRESSION	EXP_	
FLATTEN	FLTN_	Big data operator
JAGGED	JGD_	Big data operator
LOOKUP	LK_	
PIVOT	PVT_	
UNPIVOT	UPT_	
SPLIT	SPL_	
SORT	SRT_	
SUBQUERY FILTER	SQF_	
TABLE FUNCTION	TBF_	
JOIN	JNR_	
SET OPERATOR	UNION_%	Prefix should describe operator function
	UNION_ALL_%	
	MINUS_%	
	INTERSECT_%	

> Table 3-1 Standard ODI mapping operator naming

3.1.4.3. Standard ODI object naming

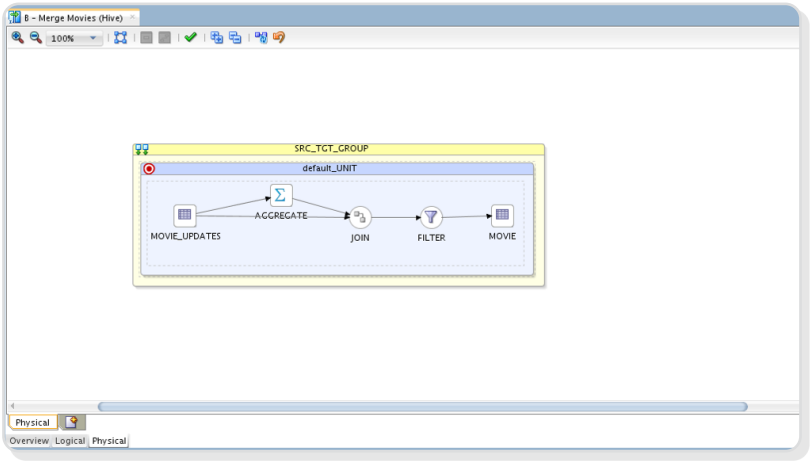
Below are examples of standard objects.



> Figure 3-5 ODI mapping

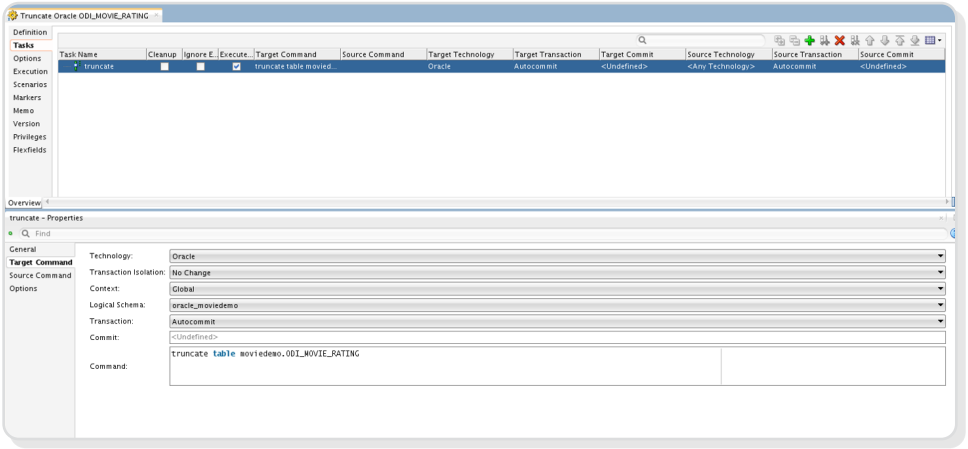
We have Logical and Physical view, in Logical view we can add elements from source to destination tables and all needed transformational objects like aggregator, filter, expression, etc.

Physical view is generated automatically, example:



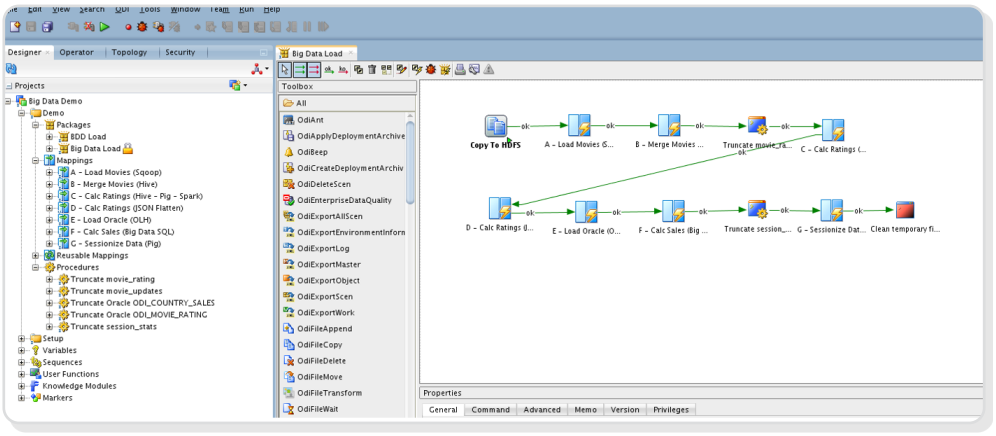
> Figure 3-6 ODI physical view

Procedure is a block of code that executes on database, when called it executes some predefined task. Example.



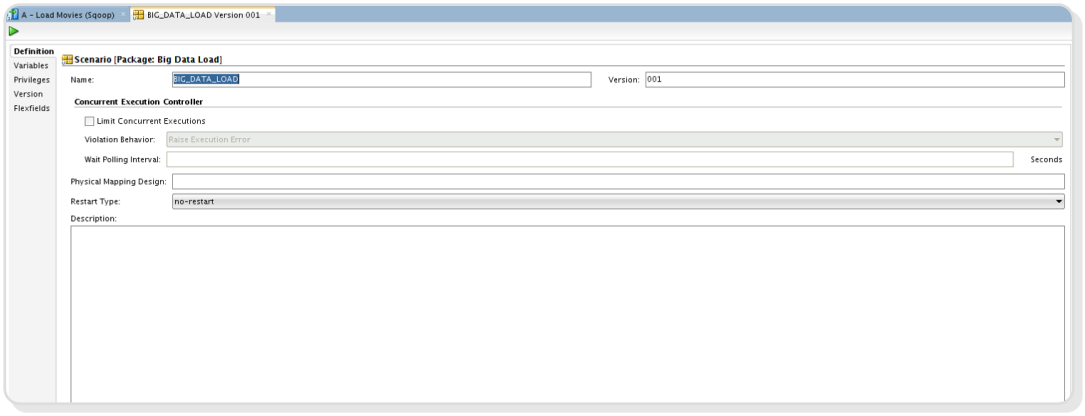
> Figure 3-7 ODI procedure

Packages can contain more mappings, procedures, sequences and it defines execution order of it's underlying objects.



> Figure 3-8 ODI package

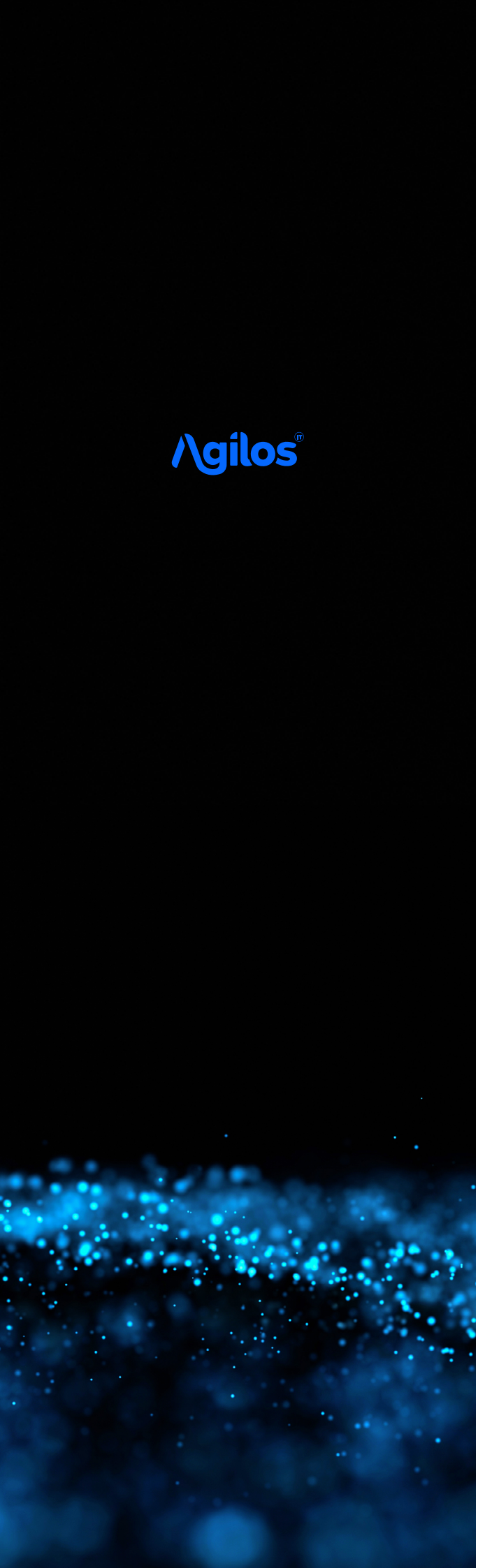
Scenario is a snapshot of mapping at given time, when we make changes on mapping, we also need to regenerate scenario.



> Figure 3-9 ODI scenario

Object	Prefix	NOTE
MAPPING	M_	IMPORT mapping
	M_STG_	STAGE mapping
	M_LOAD_	Another schema mapping
PACKAGE	PKG_	
SCENARIO	Identical to its source object	
REUSABLE MAPPING	RM_	
PROCEDURE	P_	
VARIABLE	V_	
SEQUENCE	SQ_	If a DB sequence is used inside an ODI their names must be the same.
USER FUNCTION	F_	
KNOWLEDGE MODULE	RKM_XXX_	Reverse-engineering KM
	LKM_XXX_	Loading KM
	CKM_XXX_	Check KM
	IKM_XXX_	Integration KM
	JKM_XXX_	Journalizing KM
	SKM_XXX_	Service KM
MODEL	DB schema name	
DATASTORE	Same as DB table	
LOAD PLAN	LPN_	
PROJECT	Order number – Logical unit abbreviation. For example: “02-EML”.	
FOLDER	Order number – Logical unit abbreviation. For example: “01-KLIJENTI”.	

> Table 3-2 Standard ODI object naming



4. Change management

Businesses must constantly evolve and develop new features to keep in touch with changes in technology, shifts in laws, regulations or underlying economic trends. Change management helps businesses to deploy new changes without any disruption or downtime.

Change management is the process of systematically managing changes to a software development project. It is crucial because it helps ensure that changes are properly assessed and evaluated for their impact on the project's schedule, budget, and quality. Effective change management in development requires collaboration among project stakeholders, including developers, testers, project managers, and business stakeholders.

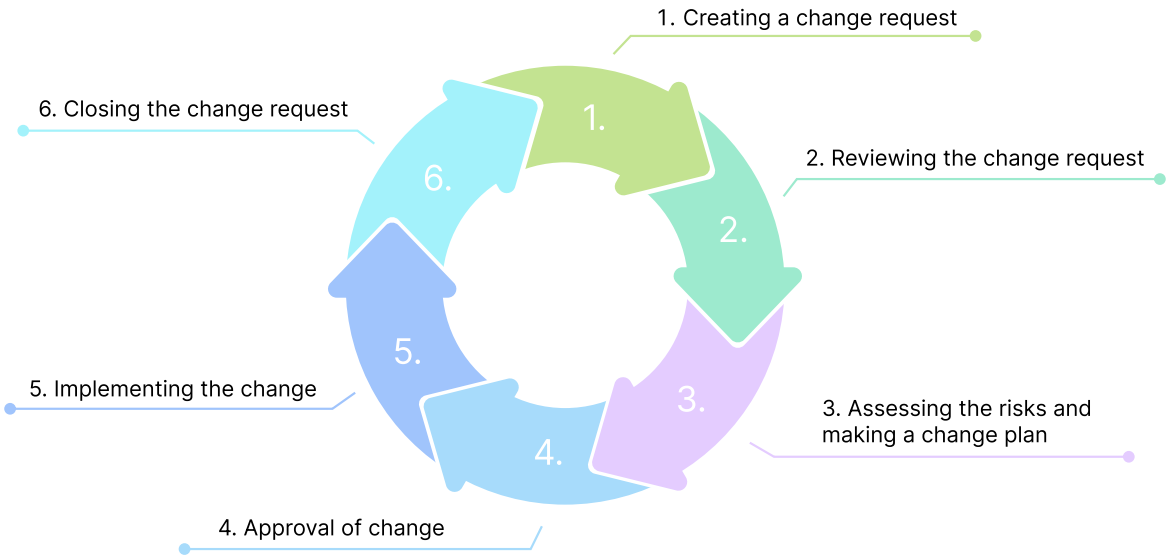
By following a structured change management process, development teams can minimize risks, avoid unnecessary delays, and ensure that the project is delivered on time, within budget, and to the desired level of quality.

Objectives of change management process:

- Reduction of risk and impact
- Maintenance of current working state
- Communication and approval from management
- Effective change planning with optimized resources
- Reduction in number of incidents due to change execution

A basic overview of a change management process is as follows:

1. **Creating a change request** - Someone requests a change and describes the requirements
2. **Reviewing the change request** - The initial change request is reviewed by a change manager or peer reviewer. What is the likelihood of success? Are the risks and benefits accurate? Is it worth the effort?
3. **Assessing the risks and making a change plan** - A game plan for the change is developed by the team - they document the expected outcomes, resources, timeline, requirements for testing, and ways to roll back the change if necessary.
4. **Approval of change** - The change is approved after being reviewed by the appropriate change manager.
5. **Implementing the change** -The team makes the change, documenting procedures and results along the way.
6. **Closing the change request** - The change manager reviews and closes the change when appropriate. They should state in their report whether the change was carried out successfully, on time, within budget, etc.

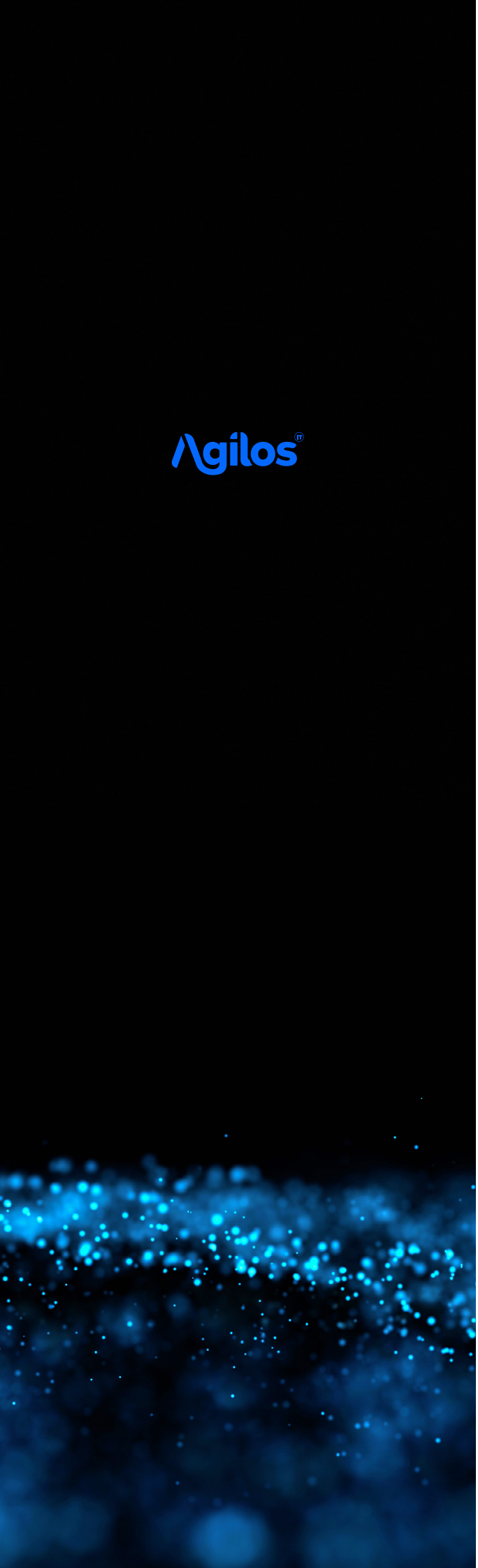


> Figure 4-1 Change management

Some best practices for modern change management:

- Learn about your company's risk tolerance and regulatory obligations.
- Whenever possible, simplify and automate change management processes.
- Accept practices that make routine change the new normal.
- Make collaboration a priority.
- Minimize your risk by utilizing chaos engineering.
- Make it easier for developers and IT teams to process change requests.
- Utilize KPIs and change metrics to unlock learning.

Change management, as a process, is crucial for businesses in order to have a quality check and deploy new changes seamlessly. Effective change management results in risk reduction, cost optimization and faster time to market.



5. Release management

The mission of release management is to create a scalable, repeatable, and controllable process that enables the organization to effectively develop and deliver products. As a result, this will lead to more consistency and the quality of the releases will rise. Improving release quality will lead to fewer incidents and fewer incidents will lead to greater efficiency and stability. The organization will be able to cope with the frequent releases of software and hardware with release management without compromising IT stability.

A well-implemented release management model has the following benefits:

- Delivering high-quality releases which are scalable for the future
- Documented policies and procedures to ensure that all stakeholders are aware of the requirements for changes and releases
- A consistent, iterative and efficient process

Development, test and production environments are at the center of the release management lifecycle.

5.1. Release management activities

5.1.1. Release policy

The release policy outlines the scope, strategy, and standards of the release practice within the organization. Additionally, it specifies the release standards that should be adhered to, including the types of releases and their frequency:

- **Minor releases** - which are characterized by small enhancements and fixes, are done more frequently.
- **Major releases** - which entail large portions of new functionalities, are less frequent and require extensive planning.
- **Emergency releases** - which are executed in response to an incident or a significant problem that requires prompt resolution.

It also includes the release naming conventions and the relationship with other ITIL processes and CMDB to ensure the smooth operation and effective management of the release process.

5.1.2. Release planning

The planning stage is the first step in the release management and this is where the entire release is structured from start to finish. Making a robust release plan will make everyone able to stay on track and ensure that standards and requirements are met properly.

Release manager creates a workflow which the entire team and key stakeholders can use throughout a release. The workflow should provide a quick overview of the release's stages and the roles played by each team member. It should entail timelines, delivery dates, requirements and the overall scope of the project.

Once the plan is sketched out, it should be presented for review to all relevant stakeholders, including the team, product manager and high-level leaders. They will give out their feedback on any deficiencies or issues they observe in the scope or requirements. Once the plan is approved and finalized, it's time for the next phase.

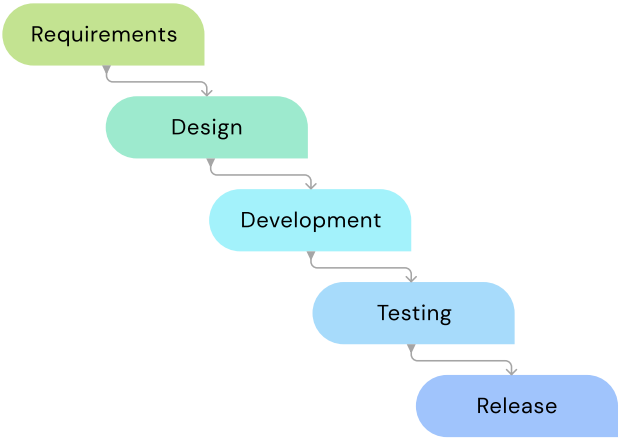
5.1.3. Design and development

The design and development phase encompasses the implementation of the software design and development in accordance with the specifications outlined in the requirements.

5.1.3.1. Development methods

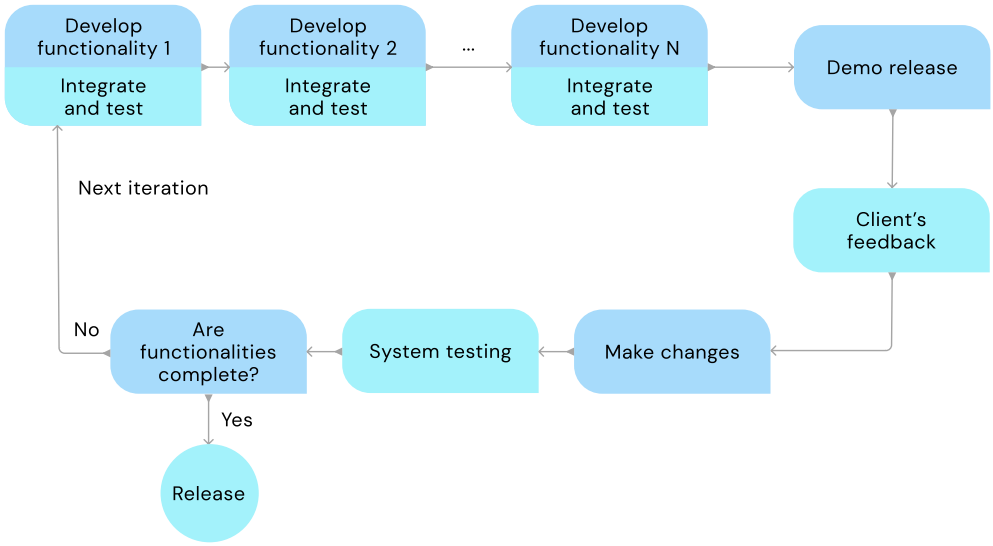
The primary objective of change management is to minimize risk, which aligns well with the sequential waterfall method. The agile methodology, characterized by its iterative and incremental delivery of individual software components, can also be integrated with the ITIL release management model with some modifications. Ultimately, ITIL advocates for the utilization of best practices that are most appropriate for the organization and its specific needs.

- **Waterfall** – delivering new functionalities every two months or quarterly



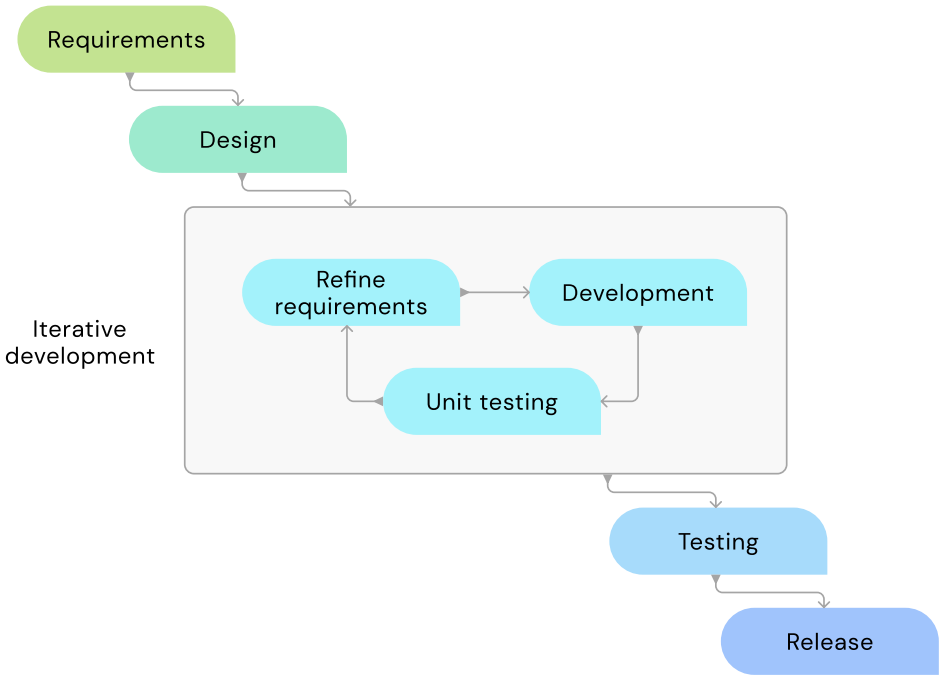
> Figure 5-1 Waterfall method

- **Agile** – delivering every two weeks



> Figure 5-2 Agile method

- **Hybrid model** – combination of previous two models (development and independent verification and validation is iterative while the rest of the phases in the life cycle follow a waterfall approach)



> Figure 5-3 Hybrid model

5.1.4. Build and configure the release

According to the plan created in the planning phase, the release manager can initiate the execution of the plan by engaging necessary resources from infrastructure, application, and other technical domains. This allows for the construction of the release in a controlled and orderly manner. During this phase, initial testing is conducted to assess the fundamental quality of the release.

5.1.5. Fit for purpose testing

Fit for use and fit for purpose are key measures for determining whether the release satisfies its objectives as outlined in the requirements. Essentially, these measures assess whether the release functions as intended. Therefore, a direct correlation between the business requirements and the new functionality should be established. Another method of determining whether the release meets the expectations of the stakeholders is user acceptance testing.

5.1.6. Release acceptance testing

5.1.6.1. Quality Assurance (QA)

To ensure the provision of a high-quality, production-ready product that aligns with the business requirements, the release must go through a structured quality assurance (QA) process. This process should consist of performance testing and user acceptance testing (UAT). The testing process may identify bugs within the release, in which case, the project team must evaluate and resolve these issues, either through bug fixes or workarounds. Ultimately, the QA team will produce a QA Acceptance Report, in which they either approve the release or decline it based on identified bugs that could cause problems in production.

5.1.6.2. Operational readiness (OR)

Operational readiness (OR) is the final step prior to implementing the release into production. The objective of OR is to guarantee that all aspects of the release have been tested, reviewed and documented. This documentation includes release notes, any known bugs and their corresponding workarounds, service level documentation, training and implementation plans and technical operational readiness testing (ORT). ORT is a thorough process which verifies that the infrastructure, network and databases operate in accordance with the business requirements. It also confirms that monitoring tools are in place and that all modifications to services and Configuration Items (CIs) have been incorporated or updated in the Configuration Management Database (CMDB).

5.1.7. Roll-out planning

Roll-out planning is a crucial aspect in ensuring the stability of all applications and systems development by adhering to a consistent and repeatable planning process. This procedure guarantees that the necessary resources are available, that detailed testing has been conducted, and that documentation has been prepared. A high-level plan is essential for every release and should encompass all test plans, acceptance criteria, and release deliverables.

5.1.8. Communication, preparation and training

The release manager is responsible for making sure that the following is included with the release:

- Support and escalation plan – document which describes what to do in case of any issues
- Training plan – document explaining usage of new functionalities to users
- Implementation plan – document explaining specifics regarding the implementation (dependencies, order of execution...)
- Communication plan – document which describes how to inform all stakeholders of all possible issues relevant to the release

5.1.9. Distribution and installation

5.1.9.1. Implementation

Release management works together with change management to ensure that the necessary steps for effective implementation are completed before obtaining approval to implement the changes. These steps (plans outlined under Communication, preparation and training) are documented within the Request for Change (RFC) document. Once this document has been completed and approved by the Change Advisory Board (CAB), the release can be deployed to production in accordance with the implementation plan.

5.1.9.2. Post-implementation

The primary objective of the post-implementation phase is to finalize the project. During this phase, all project documentation is collected, evaluated, and archived. A post-implementation review meeting may also be conducted (if there were any significant issues with any of the release management activities) to gather insights and acquire knowledge to enhance future releases.

Another important thing during this phase is closely monitoring the new service packages to make sure that they meet the service level goals that were set during the design phase. If the new service fails to meet its service level goals, it will need to be updated to improve its performance and it will go through the whole process again. In addition, it is necessary to monitor the value of the service to the clients and the business. There may come a time when the service is no longer relevant to the company's corporate strategy or no longer adds value to the business. In that case, the service will be retired.

5.1.10. Version control

Version control and source control, also referred to as revision control, is the process of managing modifications to documents, programs, large websites, and other information that is stored as computer files. It is commonly utilized in software development where a team of individuals may modify the same files. These changes are typically identified by a number or letter code, known as a "revision". Each revision is linked to a timestamp and the individual responsible for the modification. Revisions can be compared, restored, and, with certain file types, merged. Version control systems typically operate as standalone applications but revision control features are also integrated into various types of software such as word processors like Microsoft Word, and various content management systems. Popular version control systems for software development projects include Git, SVN, Mercurial, and CVS. Software tools for revision control are vital for organizations where multiple developers are working on the same projects.

5.1.11. Roles and responsibilities

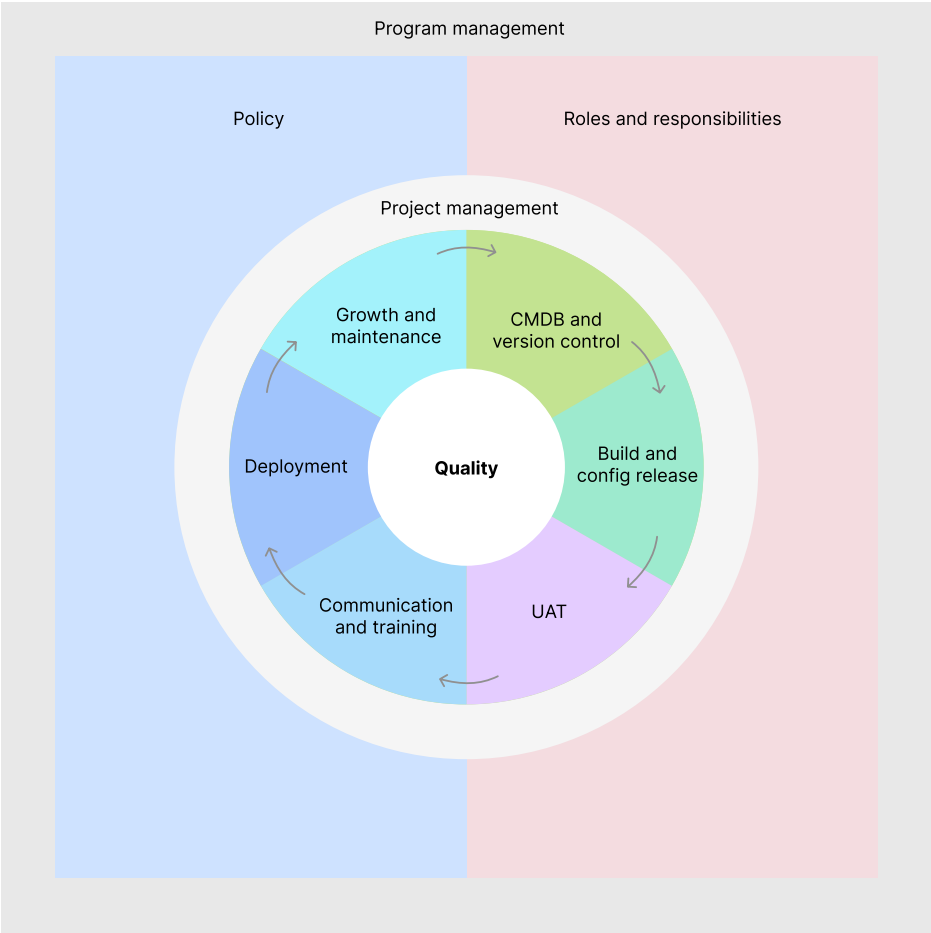
Key roles and responsibilities of a release manager are:

- Managing risks and resolving challenges that impact release scope, quality, and schedules
- Planning release windows and cycles across portfolios, components
- Communicating crucial release plans and changes
- Measuring and monitoring progress to achieve a timely software release within defined budgetary limits and defined quality standards
- Coordinating processes between different teams (possibly in various locations)
- Communicating necessary release details to business teams
- Managing, planning, and negotiating release activities
- Leading and coordinating checklist and deployment plan execution
- Deciding which automation and release management tools (or scripts) will be used for the construction, continuous integration, and deployment of software release
- Ensuring that requirements are clear across dependent project streams with an effective release
- Validating release notes

5.1.12. Release management and Project management

The main difference between these two is that the release management process can be a part of a project, while project management focuses on a project's higher dimensions. A release manager's primary responsibility is coordinating with different stakeholders for requirements, testing, and making a release calendar while taking care of possible interdependencies. Release managers ensure the synchronous running of day-to-day processes and their primary aim is to enable continuous delivery of a solution in the hands of the customers as soon as possible. Apart from that, they ensure quality benchmarks are met. A project manager takes care of resource management to make it remain within budgetary limits and quality standards. Their jobs may or may not produce a release of the projects they oversee. Project managers deliver one or more components for release that they are responsible for.

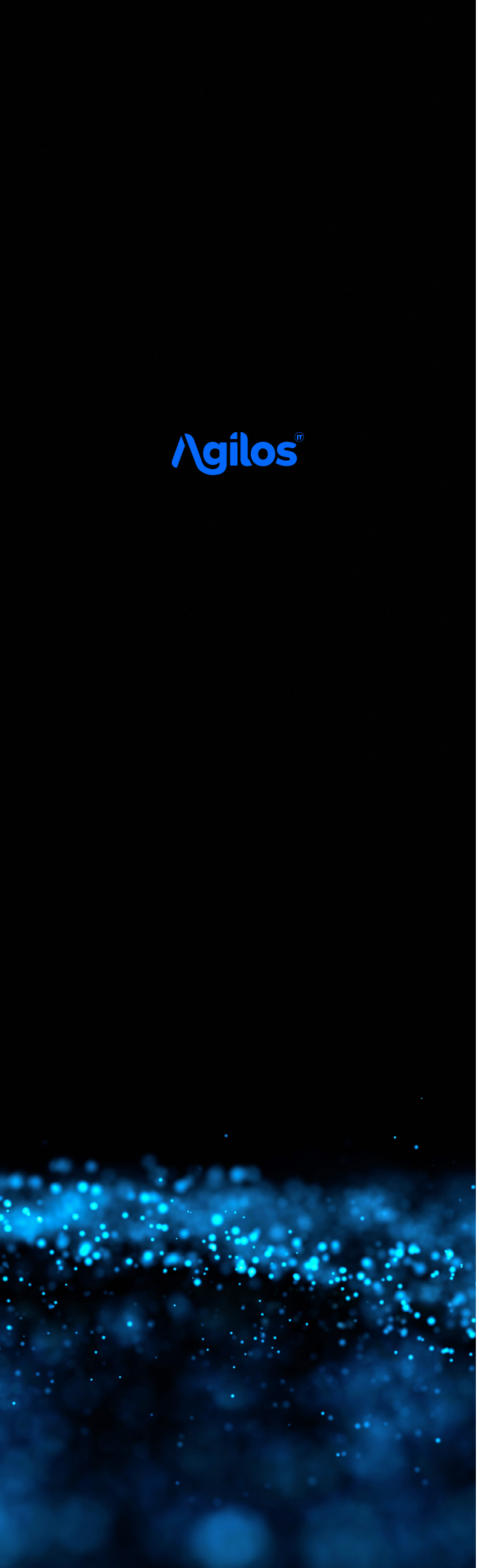
5.2. Optimal release management model



> Figure 5-4 Optimal RM model

As shown in picture, the optimal release management model consists of:

- **Program management** – there should be an overall program that manages all data warehouse releases
- **Policies** and **Roles and responsibilities** make sure everything is done compliantly and guided by role division
- **Project management** circle makes sure all planning activities are followed up and checked after completing a task in the inner circle:
  - **CMDB and Version Control -** tasks that need to be checked before configuring the release
  - **Build and Configure Release -** make sure the release follows the organization standard requirements and development processes
  - **UAT (User Acceptance Testing) -** testing tasks (like quality assurance and operational readiness testing)
  - **Communications and Training –** training users in using new features
  - **Deployment –** installation to next environment after successful testing and training; this block should also describe how to act when release needs to be rolled back
  - **Growth and Maintenance –** describes how to act on growth and how to provision servers and checks on availability in the future
- **Quality –** in the center of release management – this is what it's all about – delivering releases on scope and on schedule



6. SQL and procedural languages

SQL (Structured Query Language) is a domain-specific language designed for managing relational databases. It is a declarative language, meaning that you tell it what you want to do (e.g., “select this data” or “insert this data”), and the database management system figures out how to do it.

Procedural languages, on the other hand, are programming languages which allow you to write code that specifies how to perform a series of steps or procedures to accomplish a specific task. There are many domain-specific procedural languages designed for managing relational databases - most widely known are PL/SQL (used by Oracle databases) and T-SQL (used by Microsoft SQL Server databases).
The same as with ETL tools and ODI, we will concentrate on Oracle specific language PL/SQL in this chapter, but similar best practices can be applied to other procedural languages used for working with databases.

6.1. Structured Query Language (SQL)

6.1.1. Best practices for writing SQL queries

SQL queries are used to get data from the database. Queries can be written in multiple different ways with different levels of efficiency. Inefficient queries can take up database resources, reduce response times or even worse, make the database unresponsive. This is why it's very important to write queries in the most efficient way possible. Here are some tips for writing efficient queries.

Query formatting

Properly formatting your SQL queries is very important. It gives you greater readability which makes code overview and debugging much easier. Here are a few tips for SQL formatting:

- Each SQL command should be in a new line
- Each key command should be spelled with capital letters

Examples of bad formatting:

```
select employee_id ,first_name , last_name  from hr.employees join
hr.departments on employees.department_id = departments.department_
id where salary < 10000;
```

SQL query written above is difficult to comprehend because it's written in one line and spelled with lowercase letters. The same SQL query should be written like this:

```
SELECT employee_id, first_name, last_name
FROM hr.employees JOIN hr.departments
ON employees.department_id = departments.department_id
WHERE salary < 10000;
```

- **When using SELECT command all column names should be stated instead of ***

SELECT * command is used when it's necessary to get all of the columns from a table. Unless it's necessary to retrieve all columns, this should be avoided because it's inefficient and slows down SQL performance. Example:

```
SELECT *
FROM hr.departments;
Only the columns needed should be specified:
SELECT department_id, department_name
FROM hr.departments;
```

- **If possible, you should avoid using DISTINCT**

Keyword DISTINCT is used to eliminate duplicate results from the database. This can also significantly slow down database performance and should be avoided if possible. As an alternative you could use GROUP BY command to achieve the same result. Example:

```
SELECT DISTINCT postal_code
FROM hr.locations;
```

```
SELECT postal_code, COUNT (1)
FROM hr.locations
GROUP BY postal_code;
```

- **Using functions should be avoided if possible**

SQL functions can be very useful but they are also very inefficient because they ignore indexes which slows down database performance. If possible, using functions should be avoided. Example:

```
SELECT first_name, last_name
FROM hr.employees
WHERE substr(last_name, 1, 1) = 'K';
```

```
SELECT first_name, last_name
FROM hr.employees
WHERE last_name like 'K%';
```

The second query executes much faster because the function is omitted and indexes are used.

- **INNER JOIN should be used instead of WHERE for joining tables**

When using WHERE operator to join tables the database performs a cartesian join which joins whole tables. This can be resource demanding when working with large tables and it's more efficient to use INNER JOIN which connects only the specified rows. Example:

```
SELECT employee_id, first_name, last_name
FROM hr.employees, hr.departments
where employees.department_id = departments.department_id;
```

```
SELECT employee_id, first_name, last_name
FROM hr.employees JOIN hr.departments
ON employees.department_id = departments.department_id;
```

The second query is more readable and is written in ANSI syntax which is the preferred way of writing queries.

- **If possible, you should avoid starting LIKE statements with %**

If the % symbol is used at the beginning of a LIKE statement the database does not use indexes and instead uses a *full table scan* to fetch rows which slows down execution. LIKE statements should be carefully constructed and if only rows starting with a certain letter are needed it's important to place the % symbol in the right place. Example:

```
SELECT first_name, last_name
FROM hr.employees
WHERE last_name like '%K%';
```

```
SELECT first_name, last_name
FROM hr.employees
WHERE last_name like 'K%';
```

The first query uses *full table scan* to fetch rows while the second one uses indexes and executes much faster.

- **EXISTS/NOT EXISTS should be used instead of OUTER JOIN**

It's proven that using EXISTS/NOT EXISTS instead of LEFT JOIN with WHERE clause in queries yields better execution speeds. Example:

```
SELECT departments.department_id, departments.department_name
FROM hr.departments LEFT JOIN hr.employees on departments.
department_id = employees.department_id
WHERE employees.department_id is NULL;
```

The same query using the NOT EXISTS clause runs faster:

```
SELECT departments.department_id, departments.department_name
FROM hr.departments
WHERE NOT EXISTS (SELECT 1 FROM hr.employees WHERE departments.
department_id = employees.department_id);
```

- **OR operator should be avoided when filtering results**

OR operator should be avoided when combining more where clauses for result filtering. The UNION operator should be used instead. Databases cannot process OR operators within a single operation which is why it often slows down execution. Example:

```
SELECT first_name, last_name
FROM hr.employees
WHERE last_name like 'K%' OR last_name like 'C%';
```

```
SELECT first_name, last_name
FROM hr.employees
WHERE last_name like ‘K%’
UNION
SELECT first_name, last_name
FROM hr.employees
WHERE last_name like ‘C%’;
```

Separating the query into two separate queries using the UNION operator allows the database to use indexes and improves execution speeds.

Getting familiar with the data

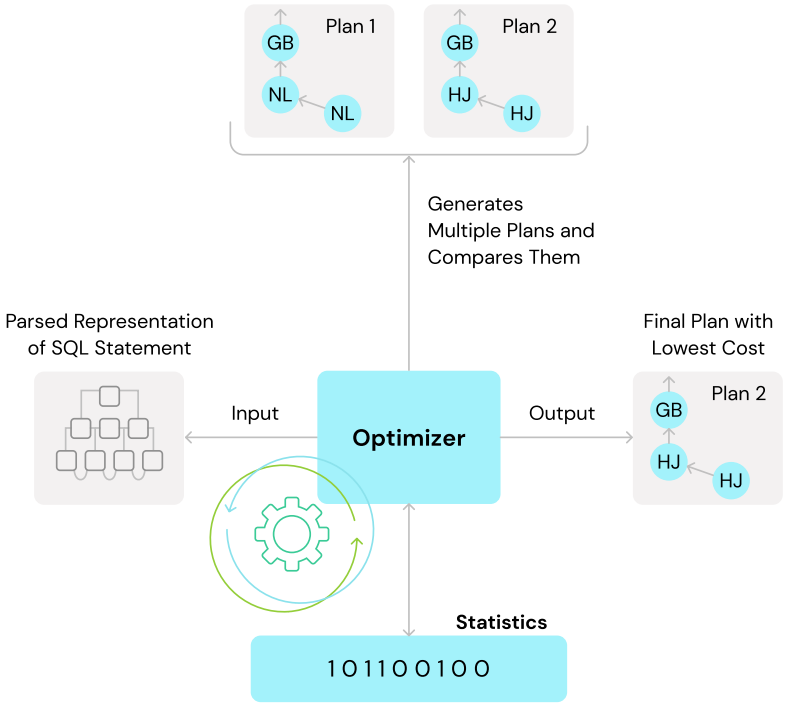
Getting familiar with the data you’re working with greatly improves your query writing ability. Knowing things like indexes, primary key columns, partitions, data and business logic can help.

Understanding SQL query execution order

It’s important to understand the execution order of your query or the way the database gets to the final result to be able to write the most efficient command. The sooner we reduce the number of selected rows from the operation set the better it’s going to execute.

1. FROM and JOIN operators fetch full referenced tables. This represents the biggest possible scope of data that can be returned. This is why you should try to reduce the possible data set as soon as possible
2. WHERE filters the data
3. GROUP BY aggregates the data
4. HAVING filters out unwanted data from the aggregated data
5. SELECT fetches specified columns
6. SET operators (UNION, MINUS, INTERSECT (ALL)) combine the results of component queries
7. ORDER BY sorts data

It’s possible that the database administrator suggests different execution order in some specific cases but the most commonly used is the one stated above.



> Figure 6-1 Optimizer

6.1.2. Execution plans

When we write SQL and want to execute it, what happens behind the scenes is that the query optimizer creates steps to execute your query against the database.

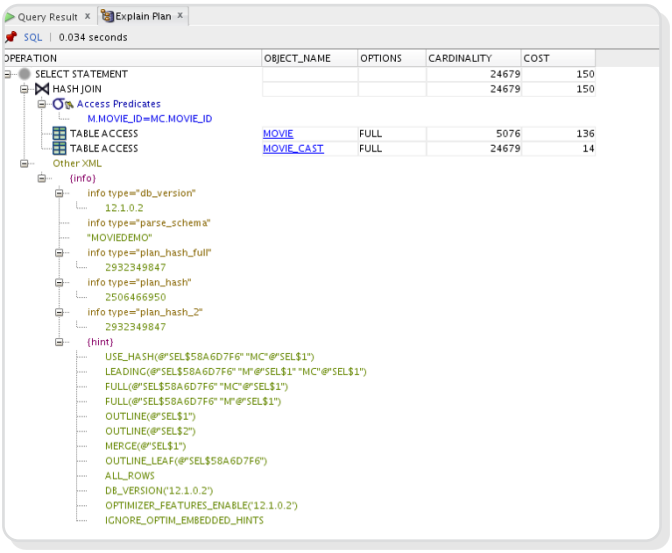
Basically, the optimizer's job is to take SQL statements and decide how to fetch the data in the quickest manner.

Before running a query, we want to make sure that we have gathered statistical information about all underlying objects. This can have a significant impact on query execution speed.

Optimizer statistics can be collected automatically by the database or they can be collected manually by using the DBMS_STATS package.

Typically, the output of the EXPLAIN PLAN statement is stored in a table called PLAN TABLE, which can be queried to determine the execution plan for subsequent statements.

Let's see this example of an execution plan. First, let's take a look at these main indicators of performance: Cost and Cardinality.



> Figure 6-2 Execution plan

Cardinality is the estimated number of rows the step will return. Based on statistics, the optimizer concluded that the query will return 233 rows.

Cost represents how many resources (CPU or I/O) the query will require to run (it affects on speed). Optimizer calculates a relative cost for each plan and then picks the plan with the lowest cost.

6.1.2.1. Join methods

Oracle supports three join methods:

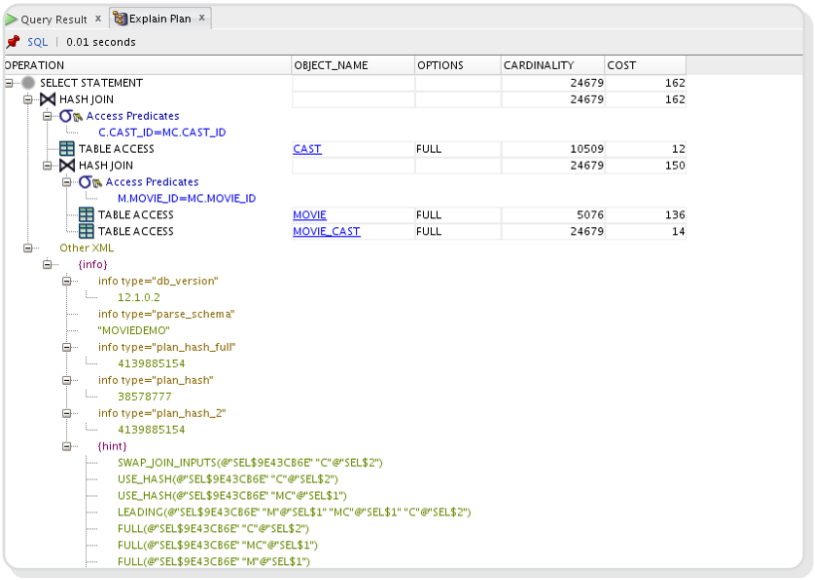
- Nested loop
- Hash join
- Merge join

6.1.2.1.1. Nested loop join

A simple nested loop join algorithm reads rows from the first table (Source table) in a loop one at a time, passing each row to a nested loop that processes the next table (Destination table) in the join. This process is repeated as many times as there remain rows to be joined. This join strategy is useful for joining two tables with small subsets of data or when one of the tables is significantly smaller in size.

6.1.2.1.2. Hash join

The hash join is similar to a nested loop join. Oracle first builds a hash table to facilitate the operation and then loops through that hash table. Oracle accesses one table (usually the smaller) and builds a hash table on the join key in memory. Then, it scans the other table in the join (usually the larger one) and searches the hash table for matches.



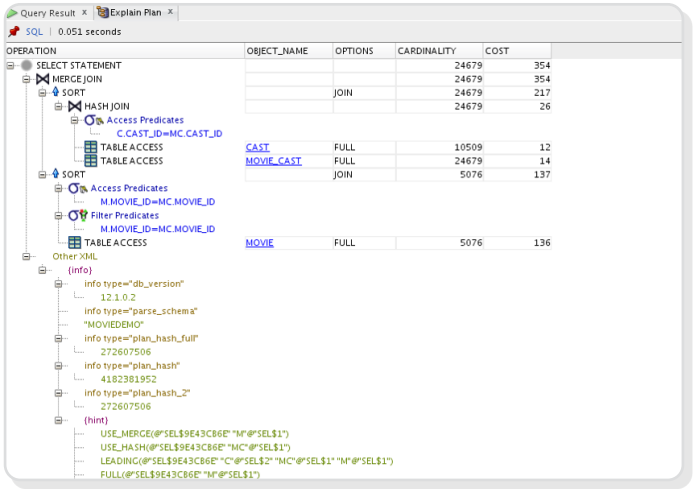
> Figure 6-3 Execution plan - hash join

TABLE ACCESS BY INDEX ROWID – means that not all information is contained in the index (columns needed are not part of the index). So, it takes the pointer to table data (rowid) and looks it up (more in “Index scans” chapter).

6.1.2.1.3. Sort-Merge Join

Sort-merge proves effective when both of the row sources are large (without the where clause or has inequality conditions such as, <, <=, >, or >=) or in the cases where there are missing indexes on join keys.

The sort merge operation is often used together with parallel query because a sort merge join always performs full-table scans and sorts data.



> Figure 6-4 Sort-merge join

6.1.2.2. Index Scans

Regardless of the index structure, an Oracle index can be thought of as a pair consisting of a symbolic key, paired with a ROWID.

Types of index access:

- the index unique scan,
- the index range scan,
- index full scans and index fast full scans.

6.1.2.2.1. Unique Scan

In index unique scan, Oracle reads index nodes and returns ROWID for the appropriate unique row from the SQL statement. An index unique scan starts processing and stops as soon as it finds the first matching record; no other matching record is possible. This makes it faster than the range scan. As the name suggests, it works only for unique index values - if we have a non-unique index, then another approach is used. Example is index on table's primary key.

```
Select ID from Customer where ID = 5111;
```

This is a fast way of accessing data because disk I/O is extremely reduced.

6.1.2.2.2. Range Scan

An index range scan is used when the SQL statement uses a non-unique index, one example would be an SQL statement that contains a restrictive clause. It returns a range of ROWID's rather than only one.

```
Select
  Id,
   customer_name from
   Customer
where
   customer_name = 'Cust1';
```

Where Customer_name is part of an index.

6.1.2.2.3. Index full scans and index fast full scans

Fast full scan is a very fast way of data access because there is no need to access table data, all the data needed is located in the index. Index fast full scan reads all of the data blocks in the index, in data block order, and index full scan does not read all of the blocks in an index. Although it is a fast way of data access, it comes with some requirements that need to be satisfied in order to be used.

Some of them are:

- the index must contain all columns needed for the query
- at least one column in the index key has NOT NULL constraint
- a row containing all nulls must not appear in the query result set

Example:

```
Select count(*) cnt, customer_name from Customer
Group by customer_name
;
```

6.1.2.3. Adaptive plans

Optimizer can change the execution plan “on the fly”, once there is a change in underlying tables (more data is loaded or deleted), Oracle changes execution plan (for example, switches from merge join to hash join because some data was deleted and index was added). That is why execution plans can differ before and after the execution of sql. This feature is called Adaptive plans.

6.1.2.4. Hints

There are cases when the Oracle optimizer doesn't get things right and executions are suboptimal, taking too much resources and time. In those cases, we need to use custom hints to enhance performance.

Mostly used hints are:

- **Append**

```
/*+APPEND*/
```

This hint is used when we want data to be inserted quickly, without additional checks. The new records are inserted above the table's high watermark (HWM). It is also called **direct path insert.**

- **Parallel**

The parallel hint is used if we are required to retrieve data as fast as possible by using multiple threads. We can specify the degree of parallelism (number of threads to be used). This hint should be used carefully because it consumes a lot of server resources.

```
Select /*+ parallel(2)*/ First_name, Last_name from Table1;
```

- **Index Hints**
  - **Index**

If we've noticed that there is an existing index on the table and that the optimizer missed to use it, we can try and force the usage of that index by adding this line after the keyword "SELECT":

```
/*+ INDEX (Employee IDX_Employee_xxx*/
```

  - **No_index**

Also, there is another case, when we don't want to use the index because of performance issues, then we use the no_index hint.

```
/*+ NO_INDEX(t1 id1) */
```

- **Nested loop**

  If we think that using nested loops while joining tables could improve performance, we can use this hint:

```
/*+ USE_NL(t1 t2) */
```

- **Merge**

  When we want to force an optimizer to **join** tables with “SORT” and “**MERGE**” operations.

```
USE_MERGE(t1)
```

- **Full**

```
/*+ FULL(t1)  */
```

  The full hint is used when we want to force the optimizer to use full table scan.

- **Use hash**

  Used when we want to enforce join hash method of two tables

```
/*+ use_hash(t1,t2) parallel(e, 4) parallel(b, 4) */
```

It is important to mention that we need to be careful when using hints because we can make execution even slower - it is recommended only for experienced developers. Most of the time, the optimizer uses the best strategy and the only thing we need to do is to check if the statistics are up to date.

6.2. PL/SQL (procedural language extension to Structured Query Language)

6.2.1. PL/SQL best practices

1. **Optimize SQL in PL/SQL programs**

   - You can take advantage of PL/SQL-specific enhancements for SQL like BULK COLLECT and FORALL, cursor variables and table functions. Performance with multi-row SQL operations is improved by an order of magnitude because much less overhead is used for context switching.
   - Hide your SQL statements behind a procedural interface so that you can easily change and upgrade. Avoid repetition and dispersion.
   - We need to assume change with data structures is going to happen and we need to build that assumption into our code with anchored declarations. %TYPE for scalar structures and %ROWTYPE for composite structures.
   - Write readable, maintainable code. PL/SQL allows you to write very readable, self-documenting and easily maintained code. This should be a primary objective for any program.

2. **Handle PL/SQL compile-time warnings and PL/SQL runtime errors in your programs**

   - Whenever you log an error, capture the call stack, error code, error stack, and error backtrace.
   - Send an application-specific error message with RAISE_APPLICATION_ERROR
   - re-raise an exception in your exception handler because the outer block doesn't know an error has occurred

3. **Comment your code**

   Structured and orderly written code greatly aids in reading and understanding the code, however, an excellent programmer also uses a lot of comments in their code. It can be straightforward to see what the code does, but understanding the intention behind the code can be challenging if it is not properly formatted and commented on. When writing comments, approach it as if you were to go through the code with a junior who has basic technical knowledge but almost no business knowledge. In such a situation, the focus is more on explaining why the program works and not how - you should explain the business need for the code, why a certain piece of code is written one way while another way is possible, why is the

following piece of code important, why a certain part is not optimal, what is the most sensitive part of the code, etc. Reading this text might lead you to the conclusion that there are not too many comments but the number of comments is a matter of perspective, there can be too few or too many. The purpose of comments is not to describe the code, because that's what the code is for. Comments should be used to explain why the code is written in a certain way.

When writing comments, a good idea would be to use a tool like PLDoc.

PLDoc (Programming Language Documentation) is a tool that generates documentation from code comments written in a specific format. It is often used instead of plain comments for several reasons:

- Consistency: PLDoc allows for a standardized format for documenting code, which can improve the consistency and quality of the documentation.
- Automated Generation: PLDoc automatically generates documentation from the code comments, reducing the amount of manual effort required to maintain up-to-date documentation.
- Improved Readability: PLDoc documentation is often easier to read and navigate, as it is generated in a structured format such as HTML or PDF.
- Increased Discoverability: PLDoc makes it easier to find information about specific code elements, as the documentation is linked to the code and organized in a logical manner.

Overall, PLDoc provides a way to generate high-quality, up-to-date documentation from code comments, making it a useful tool for improving the maintainability and understandability of software projects.

There are some rules which you have to follow in order to provide PLDoc comments:

- The comment text will be treated as HTML. You can use HTML tags, formatting, links etc.
- The first sentence of the comment (ending with a dot) becomes the "summary" for the comment, shown in the Summary part of the generated documentation.
- All tags (if any) must be placed after the main comment text.

PLDoc accepts the following two comment types:

```
/* Checking the partner record whether it exists or not */
 FUNCTION check_partner(id IN VARCHAR2) RETURN NUMBER;
```

```
/** Checking the partner record whether it exists or not */
 FUNCTION check_partner(id IN VARCHAR2) RETURN NUMBER;
```

and accepts this as well:

```
--Checking the partner record whether it exists or not
 FUNCTION check_partner(id IN VARCHAR2) RETURN NUMBER;
```

but doesn't accept this:

```
--
 --Checking the partner record whether it exists or not
 --
 FUNCTION check_partner(id IN VARCHAR2) RETURN NUMBER;
```

Here are the most important notation conventions:

- **Paragraphs**
  - Paragraphs are separated by a blank line.
- **General lists**
  - Bullet lists (HTML <ul>)
    - Using * symbol

```
* bullet list item
```

  - Numbered lists (HTML <ol>)
    - Any number from 1..9 is allowed, which allows for proper numbering in the source. Actual numbers in the HTML or LaTeX however are re-generated, starting at 1.

```
1. numbered list item
```

- Description lists (HTML <dl>)

```
$ description list item 1
 : description list item 2
```

- **Tables**

  - A table-row is started by a | sign and the cells are separated by the same character. The last cell must be ended with |. Multiple lines that parse into a table-row together form a table. Example:

```
        | Algorithm    | Time (sec) |
        | Depth first  | 1.0        |
        | Breadth first | 0.7       |
        | A*           | 0.3        |
```

- **Code blocks**

  - **Fenced**

    - The block is preceded and followed by a fence line. The traditional PlDoc fence line is ==.

```
==
   small(X) :-
      X < 2.
   ==
```

  - **Indented**

    - The block must be indented between 4 and 8 characters, relative to the indentation of the last preceding non-blank line.

- **Line breaks**

  - A line break may be added by *ending* the physical line with the HTML line break,
 or
.

For further information about using PLDoc, please refer to the user guide.

4. **Write headers for your procedures**, one nice example is below

```
/****************************************************************************************************
Procedure:          dbo.usp_DoSomeStuff
Create Date:        2018-01-25
Author:             Joe Expert
Description:        Verbose description of what the query does goes here. Be specific and don't be
                    afraid to say too much. More is better, than less, every single time. Think about
                    "what, when, where, how and why" when authoring a description.
Call by:            [schema.usp_ProcThatCallsThis]
                    [Application Name]
                    [Job]
                    [PLC/Interface]
Affected table(s):  [schema.TableModifiedByProc1]
                    [schema.TableModifiedByProc2]
Used By:            Functional Area this is use in, for example, Payroll, Accounting, Finance
Parameter(s):       @param1 - description and usage
                    @param2 - description and usage
Usage:              EXEC dbo.usp_DoSomeStuff
                        @param1 = 1,
                        @param2 = 3,
                        @param3 = 2
                    Additional notes or caveats about this object, like where it can and cannot be
run, or
                    gotchas to watch for when using it.
****************************************************************************************************
SUMMARY OF CHANGES
Date(yyyy-mm-dd)    Author              Comments
------------------- ------------------- ------------------------------------------------------------
2018-04-27          John Usdaworkhur    Move Z <-> X was done in a single step. Warehouse does not
                                        allow this. Converted to two step process.
                                        Z <-> 7 <-> X
                                            1) move class Z to class 7
                                            2) move class 7 to class X

2022-03-22          Maan Widaplan       General formatting and added header information.
2022-03-22          Maan Widaplan       Added logic to automatically Move G <-> H after 12 months.
****************************************************************************************************/
```

5. **Create UNIT tests for your PL/SQL code.**

   We recommend using utPLSQL. utPLSQL is a Unit Testing framework for Oracle PL/SQL. The framework follows industry standards and best patterns of modern Unit Testing frameworks like JUnit and RSpec.

   Best Practices from utPLSQL website:

   - Test Isolation and Dependency
     - Tests should not depend on a specific order to run
     - Tests should not depend on other tests to execute
     - Tests should not depend on specific database state, they should set up the expected state before being run
     - Tests should keep the environment unchanged post execution
   - Writing tests
     - Tests should not mimic / duplicate the logic of tested code
     - Tests should contain zero logic (or as close to zero as possible)
     - The 3A rule:
       - Arrange (setup inputs/data/environment for the tested code)
       - Act (execute code under test)
       - Assert (validate the outcomes of the execution)
     - Each tested procedure/function/trigger (code block) should have more than one test
     - Each test should check only one behavior (one requirement) of the code block under test
     - Tests should be maintained as thoroughly as production code
     - Every test needs to be built so that it can fail, tests that do not fail when needed are useless
   - Gaining value from the tests
     - Tests are only valuable if they are executed frequently, ideally with every change to the project code
     - Tests need to run very fast; the slower the tests, the longer you wait. Build tests with performance in mind (do you really need to have 10k rows to run the tests?)
     - Tests that are executed infrequently can quickly become stale and end up adding overhead rather than value. Maintain tests as you would maintain code.

- Tests that are failing need to be addressed immediately. How can you trust your tests when 139 of 1000 tests are failing for a month? Will you recognize each time that it is still the same 139 tests?

- Tests are not for production
  - Tests will generate and operate on fake data. They might insert, update and delete data. You don't want tests to run on a production database and affect real life data.

- Tests and their relationship to code under test
  - Tests and the code under the test should be in separate packages. This is a fundamental separation of responsibilities.
  - It is common for test code to be in the same schema as the tested code. This removes the need to manage privileges for the tests.

- Version Control
  - Use a version control system for your code. Don't just trust the database for code storage. This includes both the code under test, and the unit tests you develop as well. Treat the database as a target/destination for your code, not as a source of it.

6.2.2. PL/SQL naming conventions

Object type	Naming convention	Example
Parameter		
Input parameter	**p_**<*root_name*>**_i**	
Output parameter	**p_**<*root_name*>**_o**	
Input Output parameter	**p_**<*root_name*>**_io**	
Variable		
Variable declared in PL/SQL block or at procedure/ function level	**l_**<*root_name*>	
Variable declared at package level	**g_**<*root_name*>	
Constant declared in PL/SQL block or at procedure/ function level	**c_**<*root_name*>	
Constant declared at package level	**gc_**<root_name>	
Explicit cursor	[SC_]<*root_name*>**_cur**	
Cursor variables	[SC_]<*root_name*>**_cv**	
Record types	[SC_]<*root_name*>**_rt**	
Record variable	SC_<*root_name*>[**_rec**]	
Collection types		
Associative array	[SC_]<*root_name*>**_aat**	
Nested table	[SC_]<*root_name*>**_nt**	
Variable sized array	[SC_]<*root_name*>**_vat**	
Collection variables	[SC_]<*root_name*>	
Object type **User-defined type**		
Object type	**t_**<*root_name*>	
Object table	**t_**<*root_name*>**_tab**	
Exception	**e_**<*root_name*>	
Exception number	**en_**<*root_name*>	

> Table 6-1 PL/SQL naming conventions

6.2.3. Debug PL/SQL code with SQL Developer

The PL/SQL Debugger works pretty much out of the box when used with a previous Oracle version. These are the things we needed in place before we could start debugging PL/SQL:

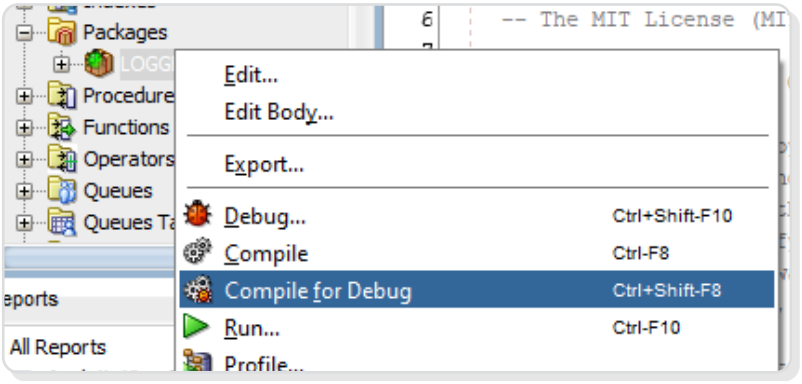
- A grant of the DEBUG CONNECT SESSION privilege
- EXECUTE privilege on DBMS_DEBUG_JDWP
- EXECUTE privilege on the stored procedure you want to debug

Starting with Oracle 12c, if you want to debug PL/SQL stored procedures in the database through a Java Debug Wire Protocol (JDWP)-based debugger, such as SQL Developer or JDeveloper, then you must be granted the jdwp ACL privilege to connect your database session to the debugger at a particular host.

This is one way you can configure network access for JDWP operations:

```
BEGIN
 DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE
 (
 host => 'localhost', -- Host can be a host name, domain name, IP address, or
subnet.
 lower_port => null,
 upper_port => null,
 ace => xs$ace_type(privilege_list => xs$name_list('jdwp'),
 principal_name => 'schema',
 principal_type => xs_acl.ptype_db)
 );
END;
```

After permissions are given, you are set for debugging. The first step is compiling for debugging. Go to the object explorer and find your PL/SQL package, procedure or function. Right click on it and select Compile for Debug.



> Figure 6-5 Compile for debug

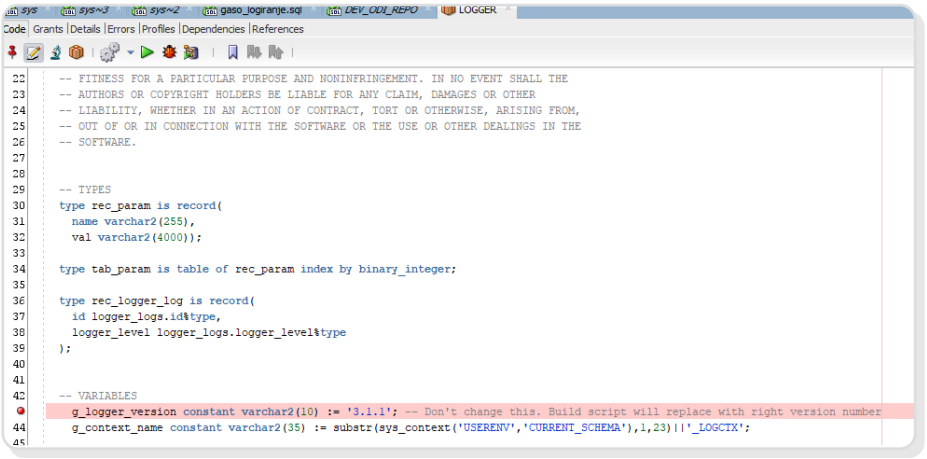
This will recompile the object and add extra information for debugging purposes. You shouldn't do this on a production server, only on a development or test server. But then again, you shouldn't be debugging on production anyway!

Once the package has been compiled successfully, you need to set up your debug session. If you start a debug now, the code will run and won't stop, unless it finds an error.

If you want it to stop, you can do it in two ways.

First, set a breakpoint. A breakpoint is a point in the code where the debugger will stop. It's useful for analyzing the path that the code has taken, as well as seeing what variables are initialized and set to.

To set a breakpoint, click in the left margin of the code on the line you want to set a breakpoint on. If done correctly, a red dot will appear in the margin.

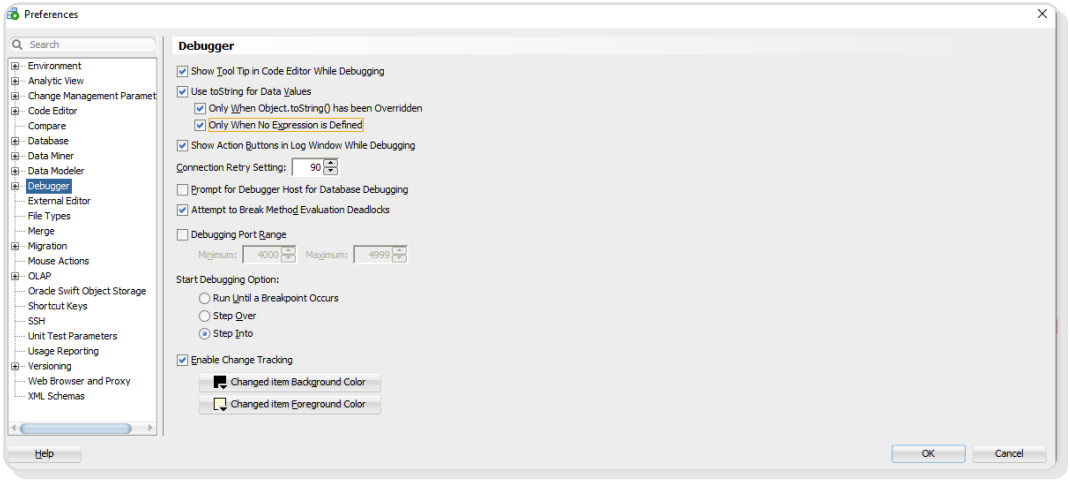


> Figure 6-6 Breakpoint

To turn off a breakpoint, click the red button again.

If you don't want to set a breakpoint, you can just run the code and step through it. However, there is no button to start the debug session by clicking Step Over, as found in other IDEs. SQL Developer's default "debug" action is to run until a breakpoint occurs.

You can change this by going to Tools > Preferences, and clicking Debugger.

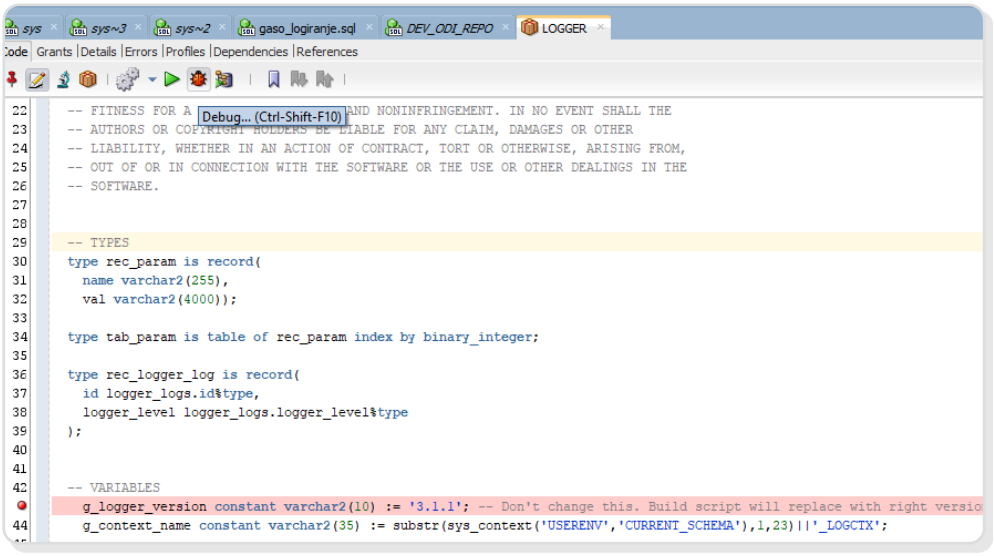


> Figure 6-7 Debugger options

Change the option that says "Start Debugging Option" to Step Into. This will allow you to click *Debug* and run to the first line of code.

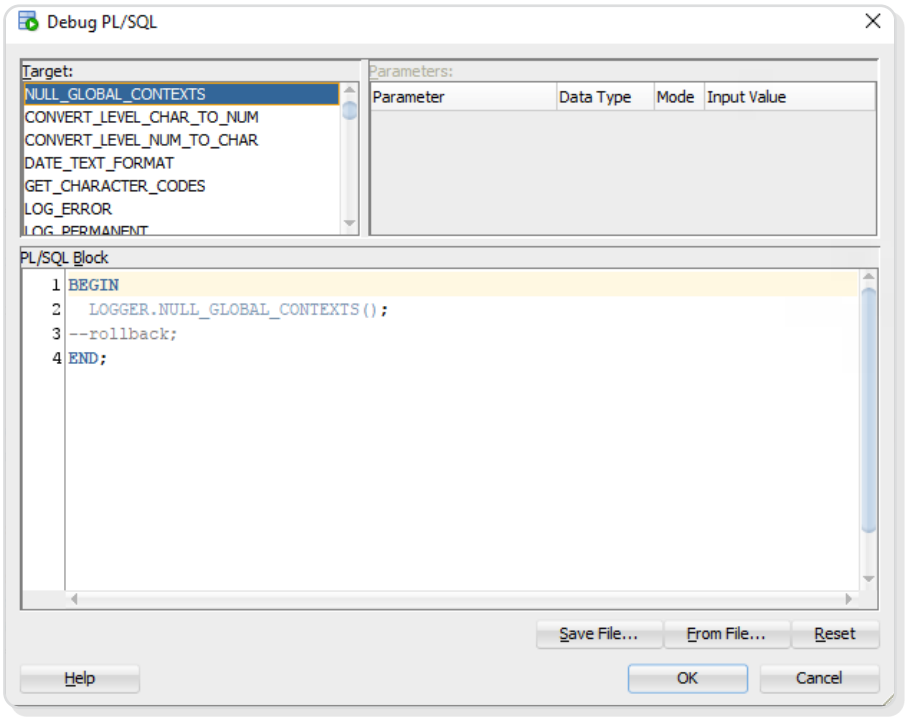
Now you've set up the environment and the code, you can start debugging

Click on the *Debug* button, which looks like a little bug.



> Figure 6-8 Debugging

A window will appear:



> Figure 6-9 Debug PL/SQL

This window basically creates an anonymous PL/SQL block of code, to run the procedure you want to debug. Here you can set up any parameters or variables before you run the code, which is very useful.

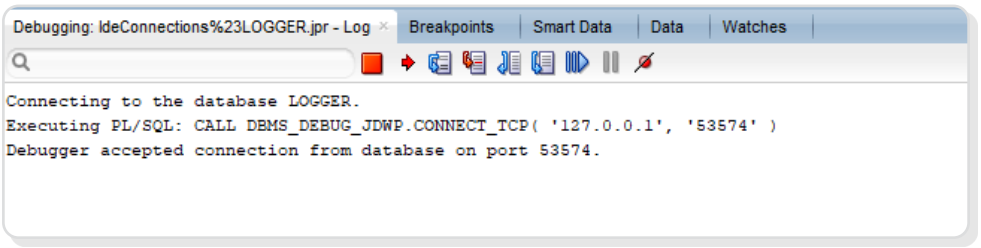
Change the code or add any parameters you like and click OK.

If Windows firewall warning appears, you can click “Allow access”.

The debugger should now be running!

There are several commands available in the SQL Developer debugger.

From left to right, starting with the big red square, they are:



> Figure 6-10 Debugger commands

- **Stop** – stops the debug session.
- **Find Execution Point** – moves your cursor to where the code has stopped.
- **Step Over** – steps over the selected line and moves to the next line in the code.
- **Step Into** – steps into the line of code selected, causing the debugger to continue inside the method or function that the line of code is currently on.
- **Step Out** – steps out of the method or function you are in, and returns to the level above.
- **Step to End of Method** – goes to the end of the method.
- **Resume** – continues debugging, until another error or breakpoint is reached.
- **Pause** – pauses the debugger in its place.
- **Suspend All Breakpoints** – turns off all breakpoints on the current database.

If you’ve used other debug tools before, then this concept may be familiar to you already.

6.2.4. Event logging on Oracle database

Logging events on the database gives us useful information when tracking process execution if the process is written in PL/SQL procedure. When designing logging, one should pay attention to code readability and simplicity. Logging should be easy to use with a small footprint.

6.2.4.1. EVT_LOG table

Event logging is using the EVT_LOG table where all information about process execution will be stored: process start, process end, process duration and an error message if the process fails to execute. Table definition with fields and descriptions is shown in the following table:

COLUMN_NAME	DATA_TYPE	NULL	RNO	COMMENT
ID	NUMBER(38,0)	No	1	Primary key of the table, using sequence to fill.
EVT_NAME	VARCHAR2(255 CHAR)	Yes	2	Name of the event: procedure, package, insert, truncate...
USER_CREATED	VARCHAR2(255 CHAR)	Yes	3	Database user that created the event.
DATETIME_START	TIMESTAMP(6)	Yes	4	Start timestamp of the event.
DATETIME_END	TIMESTAMP(6)	Yes	5	End time of the process.
DURATION_SEC	NUMBER(20,6)	Yes	6	Duration of the event or process in seconds.
TABLE_NAME	VARCHAR2(255 CHAR)	Yes	7	Table name if used id event.
ROW_COUNT	NUMBER(38,0)	Yes	8	Count of the rows affected in the process.
STATUS	VARCHAR2(10 CHAR)	Yes	9	Status of the process: ok, error.
ERROR_MESSAGE	VARCHAR2(2000 CHAR)	Yes	10	If the process fails to execute, a database error message will be written.
INFO_MESSAGE	VARCHAR2(500 CHAR)	Yes	11	Optional field to write additional information about the event or process.

> Table 6-2 EVT_LOG

6.2.4.2. LOG_EVT package

Logging is implemented in the PLS/QL package using functions and procedures.

LOG_START function

Function used to log start of the event, returns ID of the LOG_EVT table row used to pass as parameter in LOG_END, LOG_ERROR procedures.

Parameters:

- p_event_name – name of package, procedure, insert operation, truncate, delete
- p_table_name – name of the table affected in the process
- p_info – additional information about the process

LOG_END procedure

Procedure used to log end, duration, and row count of the process. End of the process is logged automatically using systimestamp, and duration is calculated using end and start time

Parameters

- p_evt_id – ID of the row returned by LOG_START function
- p_row_count – count of the rows affected by the process if known
- p_info – additional information about the process

LOG_ERROR procedure

Procedure used to log errors of the process. If the process fails to execute or an error occurs, an error message will be written about the process. Procedure is using the DBMS_UTILITY package to back-trace the error and write names of the packages and line numbers where exception is thrown.

Parameters

- p_evt_id – ID of the row returned by LOG_START function

Example of logging in the procedure:

```
procedure insert_rows(p_date in date) is
    l_log_id int;
  begin

    -- Beginning of the process logging
    l_log_id := log_evt.log_start(
      'insert_rows',
      'ACCOUNT_REPORT',
      'Insert rows into ACCOUNT_REPORT table.'
    );

    -- insert
    insert into ACCOUNT_REPORT (ID, FIRST_NAME, LAST_NAME)
    SELECT * FROM ACCOUNT_TABLE WHERE SALARY > 1000;

    -- end of the process logging
    log_evt.log_end(l_log_id, SQL%ROWCOUNT);

  exception when others then
    -- logging the error if occurs
    log_evt.log_error(l_log_id);
    raise;
  end;
```

6.2.4.3. Dependencies

Sequence

SEQ_HRAGSL_EVT_LOG – used to populate ID field in EVT_LOG table in function LOG_START

Packages

DBMS_UTILITY – used to back trace exception when error occurs in the process

PKG_UTILS – used to calculate time difference between process end and process start in seconds

Table

EVT_LOG – used to store data about process: logging table

6.2.5. Code snippets

6.2.5.1. What are snippets?

Snippets are nothing more than simple code fragments saved for later or repeated use. They can significantly improve your workflow and save time. They are often shared between team members working on the same project or even online in communities such as Oracle Apex.

Snippet operations

Following actions can be done on snippets:

- **Creating** - By clicking on Create Snippet action on the navigation bar you can create a snippet starting from scratch.
- **Creating from selection** - Select the part of the code you think is important and right-click it and save it as a snippet.
- **Sharing** - Click on the share icon to share or unshare a snippet.
- **Editing** - On the Snippets page, click the snippet name, and then click the Edit icon.
- **Deleting** - You can only delete the snippets that you own.

6.2.5.2. Snippet Contents

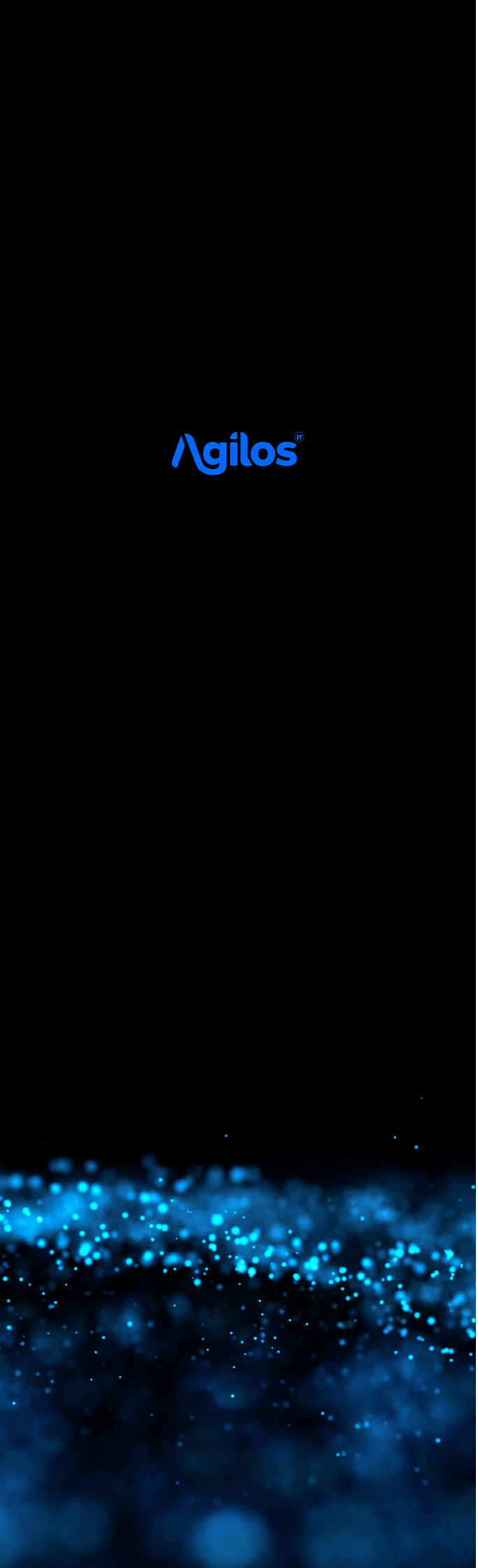
Contents of a snippet can be inserted into your SQL script either by copying it from the snippets page or by inserting it from the context menu. Adding comments to snippets is also possible and recommended especially if you are sharing snippets. Implementing GIT for version management of snippets is also possible.

6.2.5.3. Examples

Oracle SQL developer should already come with some useful built in snippets sorted into categories such as character functions, aggregate functions, conversion functions etc. Dragging and dropping one to your worksheet should insert the snippet code.

```
TRIM([LEADING | TRAILING | BOTH] trim_character FROM trim_source)
```

> Figure 6-11 Trim Snippet



7. Technical documentation

Why do we need technical documentation?

Even though it's common belief that technical documentation is used to explain what the code does, its purpose is to describe how the code does it. Main purpose of documentation is to increase product maintainability.

Technical documentation is an excellent tool for knowledge transfer. Some code might consist of complex algorithms, some of workarounds which will not be clear to every developer. Having good documentation will help in those cases.

It saves time and effort whether it's new development or an upgrade to existing code. It enables you to quickly and efficiently find relevant pieces of code.

It makes onboarding go smoother. With well written documentation, new employees will have all sorts of notes, helpful guides and directions which they can use in daily work. Instead of having to constantly ping someone on their team, they can use those resources to answer their own questions.

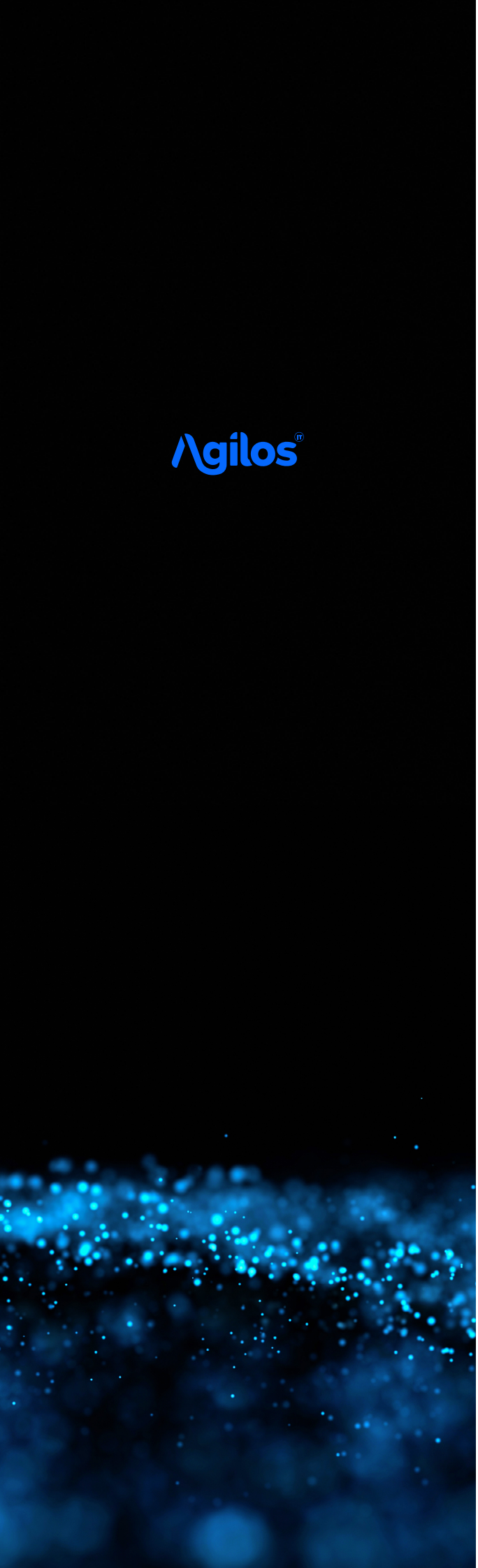
Combined with data lineage, it helps troubleshoot production issues. In case of any problems emerging after development is released, having up-to-date documentation can and will speed up fixing eventual problems.

Having no documentation is almost as bad as having too much or ill-suited documentation. Excessive documentation can lead to information overload, which in turn induces errors and impairs decision making due to having to process an excessive amount of data.

Here are some guidelines on how to write excellent documentation:

- Keep it simple and concise. Don't overcomplicate your documentation and don't use complex language. Don't comment on every line of your code; use documentation to explain something that really needs to be explained or it's not self-evident.
- Don't assume your audience is familiar with the topic; try to provide as much context as possible.
- Describe the business need for the process. The business meaning, as well as the technical one, greatly helps to understand the process.
- Don't repeat yourself. If something is already explained, point to that section, rather than explaining it again.
- Document any changes to your code. Don't just document new features, you should also document deprecating features, capturing any changes in your product.
- Review as much as you can. If possible, make sure you have several people reviewing your document.
- If you use acronyms, clearly state their meaning. Consider creating a table with acronym definitions.
- Include user manual, if needed.

Here is a sample of a template for writing technical documentation for DWH.



8. Data governance

Data governance encompasses a set of processes, standards, policies, roles and metrics which ensure the efficient and effective use of information in supporting an organization to accomplish its goals. It establishes the protocols and responsibilities which provide the quality and security of the data used across the business. Data governance defines who can take what action on what data, in which situations, and by what methods. A well-crafted data governance strategy is essential for any organization that works with data.

Data governance is a necessity in today's fast-paced and highly competitive enterprise environment. With the ability to collect vast amounts of diverse internal and external data, organizations require the discipline to maximize the value of that data, manage its risks, and decrease the cost of its management.

Compliance with relevant government regulations, such as the GDPR and the CCPA, will need to be ensured through defined retention requirements, such as a history of who changed what information and when. Data governance ensures that roles and responsibilities related to data are clearly defined and that accountability is established across the organization. A comprehensive data governance framework covers strategic, tactical, and operational roles and responsibilities.

8.1. Data cleaning

Data cleaning, also referred to as data cleansing, is the task of identifying and correcting (or removing) corrupt or inaccurate records from a dataset, table, or database. It involves identifying incomplete, incorrect or irrelevant information and then replacing, modifying, or deleting it. Data cleaning can be carried out interactively using data manipulation tools or as a batch process through scripting.

Upon completion of the cleaning process, the dataset should be in alignment with other comparable datasets within the system. The inconsistencies identified or removed may have originated from user input errors, corruption during transmission or storage, or different data dictionary definitions of similar entities in separate repositories. Data cleaning is different from data validation in a way that validation typically involves rejecting data at the time of entry, rather than on batches of data, and is performed at the point of entry rather than on existing data.

The process of data cleaning may involve eliminating typographical errors or validating and adjusting values against a predefined list of entities. The validation can be strict, such as rejecting any address that does not have a valid postal code, or with a fuzzy or approximate string matching, such as correcting records that partially match existing, known records. Some data cleaning solutions will clean data by cross-referencing it with a validated dataset. A widely used data cleaning practice is data enhancement, which involves adding related information to make the data more comprehensive. For instance, appending addresses with phone numbers related to that address. Data cleaning may also involve normalization (or harmonization) of data, which is the process of bringing together data of “varying file formats, naming conventions, and columns”, and transforming it into one cohesive dataset. An example of this is expanding abbreviations (“st, rd, etc.” to “street, road, etcetera”).

Incorrect or inconsistent data can lead to inaccurate conclusions and misdirected investments on both public and private scales. For example, the government may use population census figures to determine which regions require additional spending and investment in infrastructure and services. In this scenario, it is crucial to have access to reliable data to prevent mistaken fiscal decisions. In the business realm, inaccurate data may have a negative financial impact. Many organizations maintain customer information databases which record data such as contact information, addresses, and preferences. For example, if the addresses are inconsistent, the company may suffer the cost of re-sending mail or even losing customers.

The concept of integrity encompasses consistency, accuracy and some aspects of validation, but it is rarely used by itself in the context of data cleaning because it is not specific enough. (For example, “referential integrity” is the term used to refer to foreign key constraints.)

High-quality source data is related to the “data quality culture”” and must be established at the top level of the organization. It’s not only about implementing robust validation checks on entry screens, as users can often get around them, regardless of how strong the checks are.

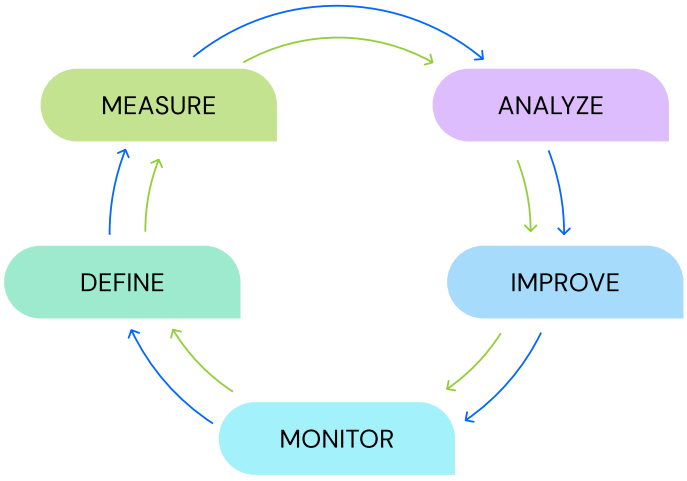
8.2. Data Quality

Data quality is evaluated from a user perspective by measuring how well it meets the project requirements. Some organizations enforce a data government policy which precisely defines expected data quality and data privacy. Data quality can also be enforced by governing or regulatory institutions like GDPR or BCBS 239. DQ circuit loop should be used to achieve a certain quality standard.

8.2.1. Data Quality Circuit Loop

A DQ circuit loop (pictured below) should be implemented to maintain the desired data quality standard.

8.2.2. Data Quality Roles



> Figure 8-1 DQ circuit loop

User roles required for successful data quality maintenance:

- **Data Owner** - Data owner is usually a senior team member with good business logic understanding and good communication skills who is responsible for data privacy and quality. It's possible to have more than one data owner although in that case only one should be responsible for the full picture data quality. Data owner controls a data domain (marketing data, controlling data etc.).
- **Data Steward** - Data stewards are typically team members under the data owner's lead. They should be responsible for day-to-day data quality operations and end user support. Data stewards should have a good understanding of both the business and technical side of data quality control.
- **Data User** - Data users are typically data warehouse development team members that are working with the data. They often set up required data quality controls. Data users should be proficient in data analysis skills and also have a deep understanding of data and business logic.

Having skilled data quality team members and well defined and widely accepted data quality roles is key to successful data quality management.

8.2.3. Defining the Rules

Defining and enforcing well thought out data quality rules and constraints should be a priority when starting the DQ process. This should be done by a team member with a great understanding of the data warehouse and its use.

8.2.4. Finding DQ Rules

The most significant input when defining data quality rules should be given by data users considering they are the ones working with the data itself. This approach should also help with the development team's acceptance of the proposed rules and controls. If data users are not familiar with the earlier layers of the data warehouse, setting up rules this way could lead to potential problems as data corruption in earlier layers can't be detected.

8.2.5. Handling Errors

Error types possible when working with data warehouse:

- **Wrong transformation logic in the data warehouse**

  Duplicates, incorrect values, “lost” data and similar are the most common data quality problems. Often the reason they happen is overly complex transformation logic as a result of the complex IT landscape.

- **Unstable load process or wrong handling of loads**

  Job orchestration errors like jobs not executing, jobs starting late or early and similar or errors due to user intervention like skipped jobs or job execution in the wrong context are the most common errors of this type.

- **Wrong data transfer of data sources**

  Errors like transferring or processing empty or incomplete data as result of mismanagement of source data.

- **Wrong operational data**

  Operational data contains errors that are not detected.

- **Misinterpretation of data**

  The data itself is not corrupted but the interpretation of it is not correct. This is a data governance issue that should be taken care of by data stewards.

These problems are often caused by people lacking the appropriate know-how and skills to define, implement, run, and work with a data warehouse solution.

8.2.6. Data Quality Dimensions

DQ dimensions should be used to sort DQ checks into groups. There can be as many of them as needed but in practice there are only a few of them that are necessary.

- **Completeness** - Checks data availability and accessibility
- **Consistency** - Checks data inconsistencies and conflicting data
- **Uniqueness** - Checks for duplicates
- **Integrity** - Checks links between data and data constraints
- **Timeliness** - Checks if all data is updated.

Data generated by the data warehouse **load process** can be helpful as well.

- **Tables with discarded data** - Data that causes errors in the loading process due to integrity constraints or other errors could be skipped or delayed by a data warehouse process.
- **Logging information** - Information about the errors could be saved into designated tables or files.
- **Bill of delivery** - Bills of delivery or summation of loaded data are sometimes generated by operational systems and can be used for reconciliation checks. They contain information like number of records, number of distinct keys, sum of values etc.

Implementation of these checks still requires analysis and error handling by a team member responsible for data quality.

The process of executing data quality rules and checks needs to be efficient and organized well enough to accommodate a large number of rules (sometimes thousands). Data stored in relational DBMS should be only partly checked because of the constraint implementations already built. Avoid DQ control if:

- Mandatory columns contain NULL
- Primary key columns contain non-unique data
- There are no existing foreign keys

It should be considered that data definition and constraints can theoretically change at any time.

Also, if the organization implementing DQ controls is large enough there should be some sort of “data quality for data quality” control and rules. Depending on the number of developers working on data quality there can often be overlapping or conflicting rules. It’s very important to keep track of and properly manage DQ controls.

8.2.7. DQ Rule Classes

Types of tests used to classify DQ rules:

- **Data quality check** - Standard checking method, one or more tables are checked on one data warehouse layer
- **Reconciliation** - Used to check completeness DQ dimension. Checks if the data was transported correctly between layers. It can be done in a single row or using a summarized approach. Reconciliation should be done between each layer because skipping layers will require more complex transformation logic
- **Monitoring** - Used to check for growing data differences between data warehouse and operational data building over time. Summarized time series of data should be implemented to check for issues like these

8.2.8. DQ Issues Quantification

After identifying issues using checks we explained before you should consider how to quantify the result data. Returning the number of rows with row IDs is not enough for data quality.

Consider adding:

- **How to quantify/count the detected errors** - Depending on the data summarization might be needed, for instance monetary values. Sometimes returning both numbers of rows with a summarization value might be the most appropriate.
- **Population** - Amount of data that's subject to specific data quality control. It allows you to represent corrupt data as a percentage of all data which is very important.
- **Definition of the result** - It's possible that some corrupt data might not cause errors which is why traffic light systems should be implemented. Each identified data point should be given a percent value and a threshold should be defined on business unit level.
- **Collect sample error rows** - Some number of found error rows should be returned to the data user for easier and more efficient examination.
- **Whitelists** - It's possible that DQ controls will find known errors that can't be corrected in the foreseeable future. These rows should be placed on a whitelist so that the data quality control mechanisms know about them and can skip them if encountered.

8.2.9. Metadata

Metadata is important to route the “Analyze” and monitor the phases of the data quality control loop.

- **Checked items** - Checked tables and fields can be assigned to DQ rules and rules can be automatically assigned to users. This is useful in instances where you need to prove how the data is checked.
- **Data user** - All DQ rules should have at least one data user assigned to check the results during the “Analyze” phase.
- **Data owner** - Every DQ rule must have a data owner assigned.

Assigning users to rules can be done automatically via data lineage. This approach can produce varying results as it allows for the assigning rules to users that are not familiar with them.

8.2.10. How to Measure Data Quality

Data quality rules should be implemented so that they don’t compromise data loading processes. Optimally, load and DQ processes should be run in parallel and be set up in a way that error reports generated make analysis phase easier without affecting load execution performance. In order to detect errors and corrupt data as soon as possible DQ processes should be run as early as possible.

8.2.11. Analysis

The final step of the process is analysis of the checked data and taking actions to correct the errors. This task should be assigned to data owners and data users who should act according to predefined methods.

Some possible actions:

- **Serious problem** - Repeat loading process after fixing the issue.
- **Problem is acceptable** - Fix the issue in loaded data and try to fix the issue for future load processes.
- **Defective DQ rule** - DQ rule should be fixed.

It’s useful to implement a data quality dashboard for easier and timely reaction to detected errors in data. For smaller teams with less resources and time this can be especially useful.

DQ dashboard should have:

- All rules assigned to a given role
- The rules' results
- A mandatory comment or a note
- An option to ignore the error on a business unit level.
- Showing rules that were not executed
- Load process status

Lastly, key performance indicators could be implemented for easy overview of DQ check results. After that, it's the data owner's responsibility to make sure all errors are noted and handled correctly.

8.2.12. DQ Processes maintenance

As the data warehouse changes, some DQ processes will also need change and get updates.

These three things should always be kept in mind:

- **Keep it up to date** - Data warehouse changes that call for changes in DQ rules should be implemented as soon as possible.
- **Enhance** - Developing new rules for errors that were not handled.
- **Streamline** - Disabling unneeded DW rules and consolidating overlapping rules.

8.2.13. DQ Processes Monitoring

To improve the DQ process it should be closely monitored over time.

Elements to monitor:

- Data coverage
- Amount of DQ findings within the active rules over time
- Number of active data quality rules
- Time needed to have all findings rated and fixed

8.2.14. Final thoughts

Closing points that should be considered:

Anticipate resistance - Unless it's explicitly demanded by compliance and regulatory demands, DQ controls will most likely be viewed as an additional workload without significant benefits for the development team.

Find a sponsor - Having a highly ranked organization member behind your back can help with establishing a DQ system.

Find allies - Like-minded team members are crucial in keeping the DQ circuit loop working properly.

Start small - If the implementation of the DQ system is just starting, the best option would be to start with a single business unit.

Don't lose sight of the whole picture - Some elements should be set in place (roles especially) no matter the scale of the project.

Once implemented, don't let go - Focus on data quality processes needs to be maintained as long as the data warehouse development is active.

Data Quality tool - Consider using an existing Data Quality tool – there are many options available, among others, DataQuality application by Agilos IT.

8.3. Data lineage

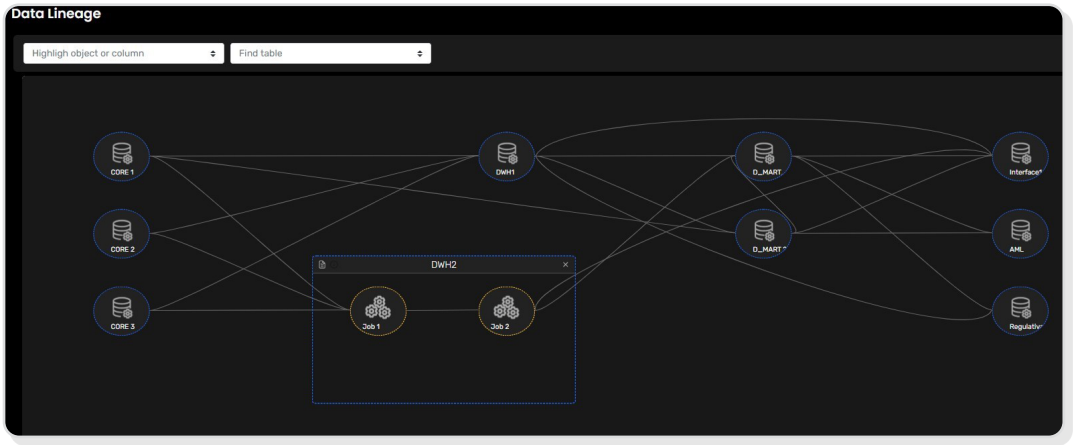
8.3.1. What is data lineage?

Data lineage traces the path of data throughout an organization's IT systems, illustrating how it flows between them and is transformed for various purposes. By utilizing metadata, which provides information about the data, it allows both end-users and data management professionals to follow the history of data assets and understand their business context or technical characteristics.

For instance, data lineage records can assist data scientists, other data analysts and business users in understanding the data they are working with and ensuring it aligns with their information needs. Data lineage also holds significant value in data governance, master data management and compliance initiatives. Among other aspects of those programs, it streamlines two vital data governance processes: identifying the underlying reasons for data quality issues and assessing the effects of changes to datasets.

Data lineage information is gathered from operational systems as data is processed and from the data warehouses and data lakes which store data sets for business intelligence and analytics applications. Along with comprehensive documentation, data flow maps and diagrams can be created to provide visualized perspectives of data lineage correlated to business processes. To ease access to lineage information for end-users, it is commonly integrated into data catalogs, which inventory data assets and the metadata associated with them.

8.3.2. Why is data lineage important?



> Figure 8-2 Screenshot from our DataLineage tool

Data lineage information is essential for data management and analytics endeavors. Data can be better managed and utilized by organizations with the assistance of lineage details. Without access to this information, it becomes challenging to fully capitalize on the potential business value of data.

Some of the advantages of data lineage are listed below.

- **More useful and accurate analytics -** By providing analytics teams and business users with information on the origin and significance of data, data lineage enhances their ability to locate the data needed for business intelligence and data science applications. This leads to more accurate analytics outcomes and increases the likelihood that data analysis will yield valuable information to inform business decisions.
- **Enhanced data governance** - Data lineage also facilitates monitoring data and carrying out other essential aspects of the governance process. It assists data governance managers and team members in ensuring that data is accurate, clean, and consistent and that it is secured, managed, and utilized appropriately.
- **Increased data security and privacy protections** - Organizations can leverage data lineage information to identify sensitive data which requires robust security. It can also be used to establish various levels of user access privileges based on security and data privacy policies and to evaluate potential data risks as part of an enterprise risk management strategy.
- **Elevated regulatory compliance** - Organizations can benefit from enhanced security measures informed by data lineage to ensure compliance with data privacy laws and other regulations. Well-documented data lineage also simplifies internal compliance audits and reporting on compliance levels.
- **Optimized data management** - In addition to data quality enhancement, data lineage improves various other data management activities such as managing data migrations, eliminating data silos, and identifying and resolving missing data in data sets.

8.4. Data security and privacy

Data warehouse security is essential and we need to design flawless systems to protect business and customer-sensitive data. Access to DWH should be managed by creating roles with different levels of access for different groups of employees. Another important thing to take care of, is making sure that sensitive information is not understandable and that the data is not personally identifiable. It is very common that data for development and test environments are taken from production databases because real data is the best to work with. But in this case, all the sensitive, personally identifiable information is available in the test or development environment. To handle this issue, different kinds of data masking methods can be used.

8.4.1. Data masking

Sensitive data which is personally identifiable needs special attention because it is vulnerable for data breaches. Data breach can happen by system error or malfunctioning, or when someone is stealing the data (either an employee who has access or an attacker from outside the organization). To lower the risk of data breaches, we should use data masking methods.

Data masking in DWH is most commonly used for creating test data because employees usually have greater access to data in test environments than they do in production. For the organization, leaked data means a loss of money and, more importantly, losing trust from clients.

Masking the data also gives a better overview of where the sensitive data is and who is using it, and having that overview should be a must have for every organization because it is important to ensure that sensitive information is stored and secured according to the laws of privacy and regulations. After a data breach happens, additional work and resources will be needed:

- Technical overview and organizational audit of why data breach happened
- Possible new development to fix the reason why the breach happened
- Explaining to customers and media the reason of data breach
- Regaining organization reputation
- Fines and legal judgements

8.4.1.1. Data masking techniques

There are different ways of masking the data, some most commonly used are:

- Shuffling – values in column are shuffled randomly
- Randomizing – generating random values
- Hiding – hiding values completely by using views
- Substitution – each character or number is replaced by another value
- Scrambling – value has some part scrambled with a symbol
- Blurring – turning a value into a certain range of values
- Encryption – value is encrypted with some encrypting algorithm using a secret key

8.4.1.2. Data masking in Oracle DB using DBMS_REDACT package

Redaction policies are a set of rules for redacting data on table level. There is a PL/SQL package DBMS_REDACT, in Oracle 12c and newer, used for creating and maintaining redaction policies on a table. Creating these policies makes data masking quick and simplifies managing the formats of masked columns as we have the possibility to mask data using different techniques. Using this method, everything is done on table level and not by creating views or doing the masking in the application. Nothing changes for the user who is allowed to see the data without the masking.

8.4.2. User permissions in DWH

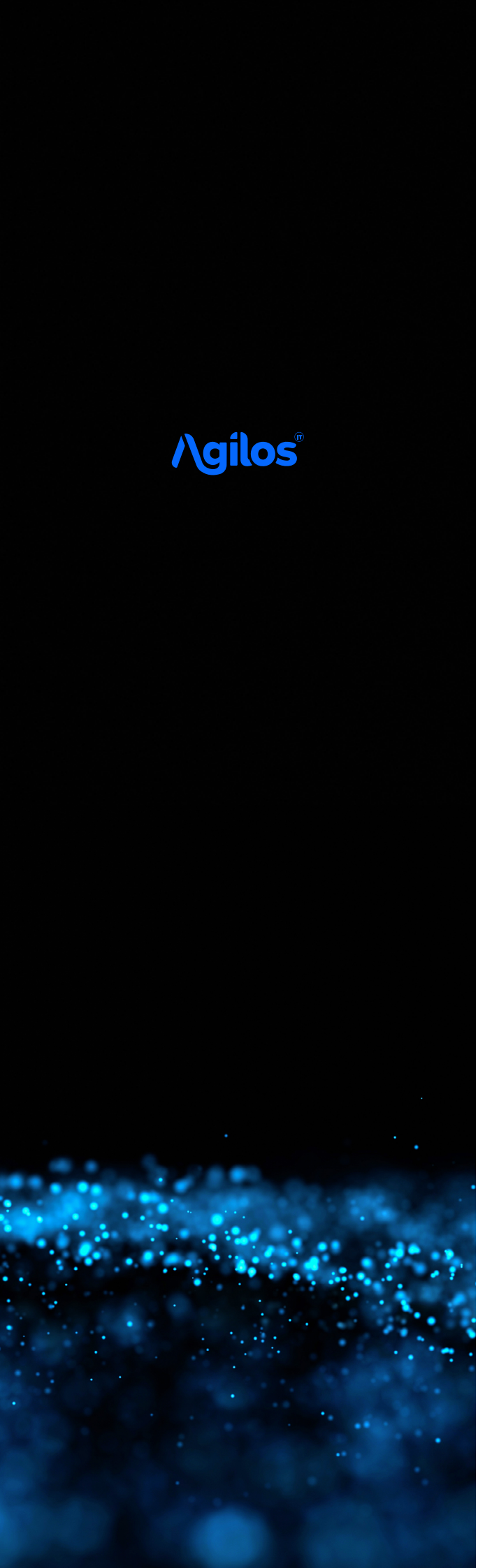
Handling user permissions in databases is usually done by creating roles for different user groups. A simple solution would be to have a layered architectural solution where a certain user would have the access to all the objects in a specific layer. Usually, that is not the case and the solution required is more complex. Analyzing users and their needs is necessary to determine how many roles should be defined and what rights should each role have.

When designing the warehouse, it's advisable to think about the data people will access and then classify both the information and end-users.

There are two data classification approaches:

- **Sensitivity-based** - highly-sensitive personal information will be more restricted while generic data will be available to more users.
- **Function-based** - specific user categories will be able to access only the data they need for their work and other information will be blocked.
- And two user classification methods:
- **Hierarchy-based** - this model is suitable for enterprises with few departments (you could create data marts with unique access for each team).
- **Role-based** - if a company has a lot of branches with the same data required, it's better to set accesses based on roles: administrators, developers, analysts, etc.

Choosing one way or combining several of them, a comprehensive, yet scalable, data warehouse architecture can be built.



9. Data pipelines

Process of data collection and analysis is called data integration and DWH is just a part of the whole process so there are other possible methods/modifications of dealing with large volumes of data.

DWH is a part of a broader concept called Data pipelines. It is the system of pipelines connecting disparate data sources, storage layers, data processing systems, analytics tools, and applications.

They are necessary for real-time analytics to support fast, data-driven decisions.

Examples of data pipelines:

- Batch based
- Data lake
- Data mesh

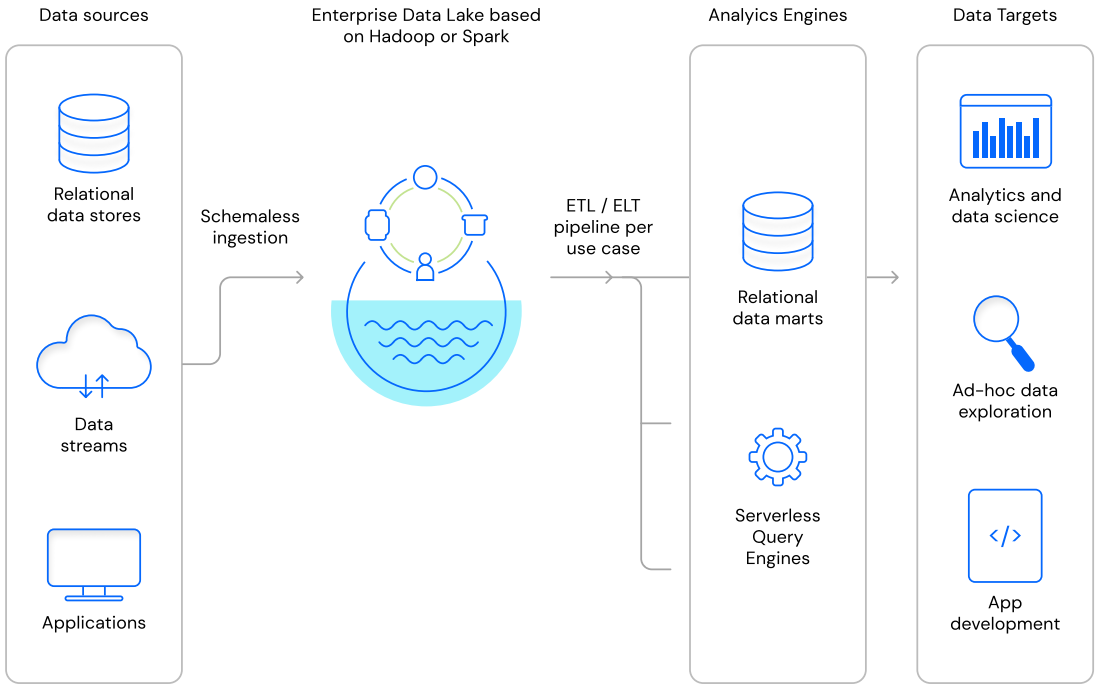
9.1. Batch based (Enterprise Data Warehouse)

DWH is one example of a batch-based data pipeline, it stores large volumes of data from applications/core systems. This is the conventional method of handling vast amounts of data.

9.2. Streaming data pipeline (Data Lake)

To reduce the engineering burden associated with data warehousing, large volumes of data from many sources may now be simply ingested and stored in an object store like Amazon S3 or on-premise Hadoop clusters. The data lake holds data in its original, unprocessed form, making it possible to store complicated and streaming data just as easily as batch files that are structured. Data can be stored in DWH (Redshift, Snowflake) or we can use serverless query engines (Athena, Starburst).

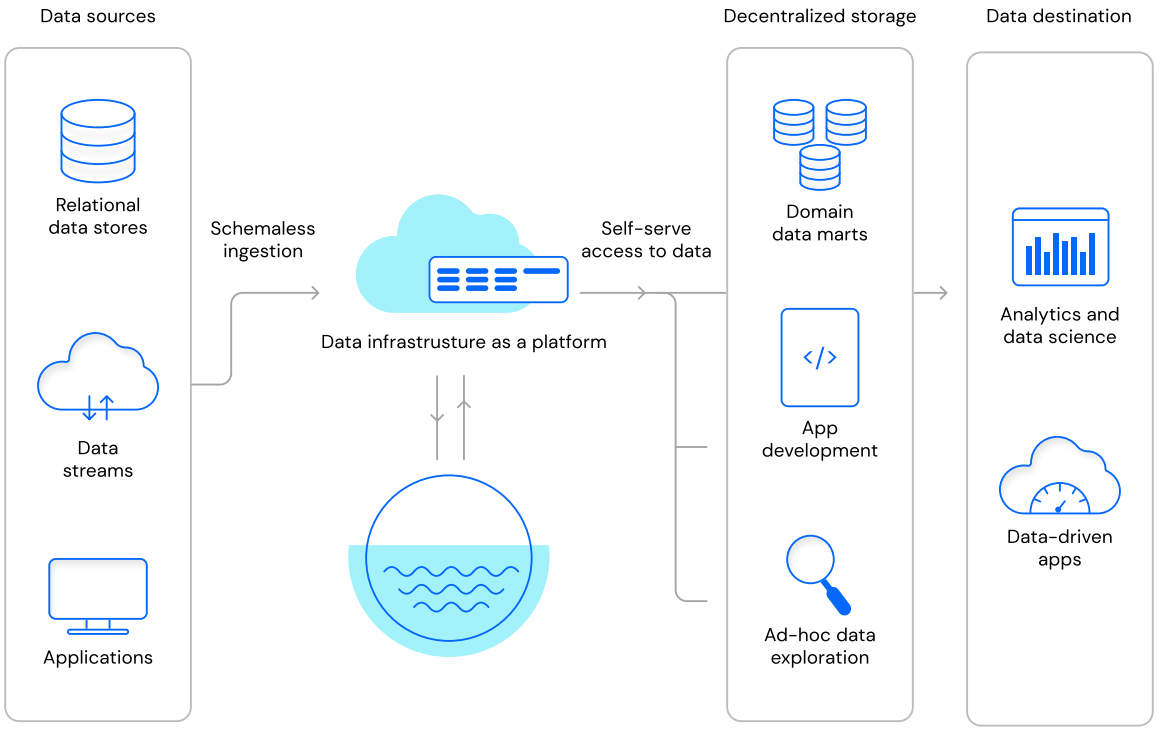
Utilizing a “save now, analyze later” strategy, this method has the advantage of enabling enterprises to manage greater volumes and more types of data.



> Figure 9-1 Data Lake

9.3. Data Mesh

Similar to the data lake technique, with this architecture raw data is fed into object storage with little to no preparation. The data can then be extracted from the lake by various teams, who can then run their own ETL or ELT pipelines to produce the dataset required for additional analysis. The data is then prepared using uniformly enforced protocols and written back to the lake in an open file format, such as Apache Parquet, while keeping important details about the data set in a business catalog. This provides the advantages of decentralized data domains while ensuring that it can be found and used by other teams, without requiring a centralized data team to handle every aspect of it.



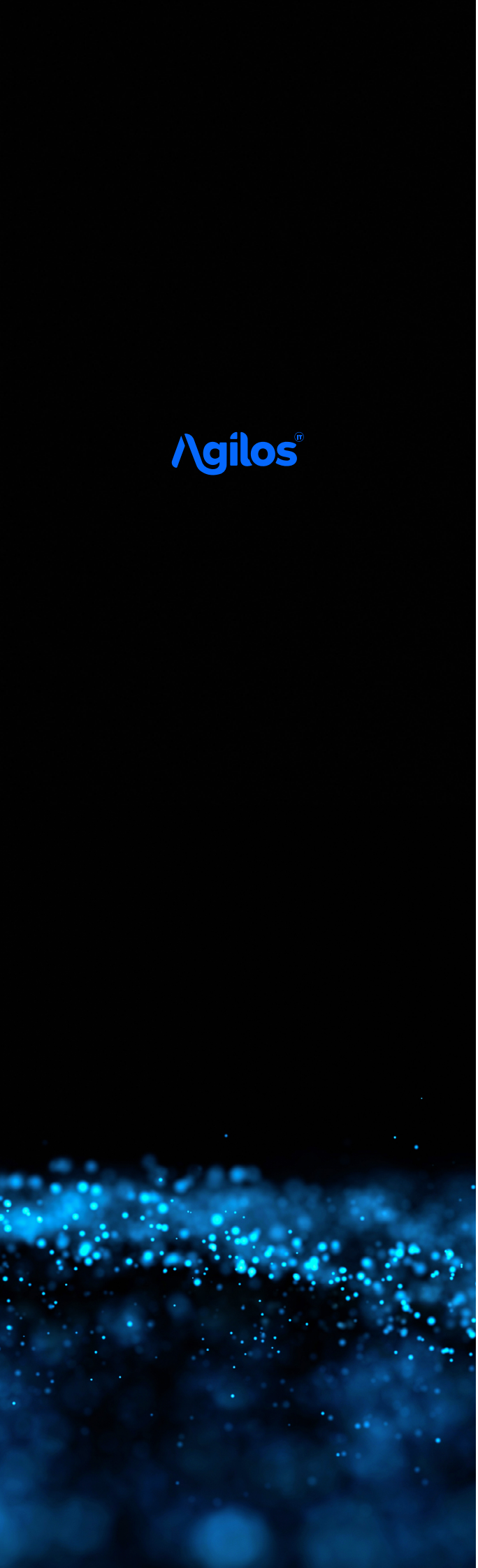
> Figure 9-2 Data Mesh

Agilos®

10. Our tools

Our company is constantly trying to optimize DWH processes by developing new tools and methods based on automation using custom built tools like SQLtoODI, Data Lineage and DataQuality. SQLtoODI is a tool which automates ODI mapping generation based on input SQL queries from users. Data Lineage is a tool which helps you visualize data flow inside data warehouses. DataQuality tool is used to guarantee data quality and make the process of data cleaning faster and easier.

These tools give us a huge functional power that dramatically speeds up big data processes and reduces human expertise necessary. This potentially reduces our users' operational costs while also reducing the time necessary to get analysis results.

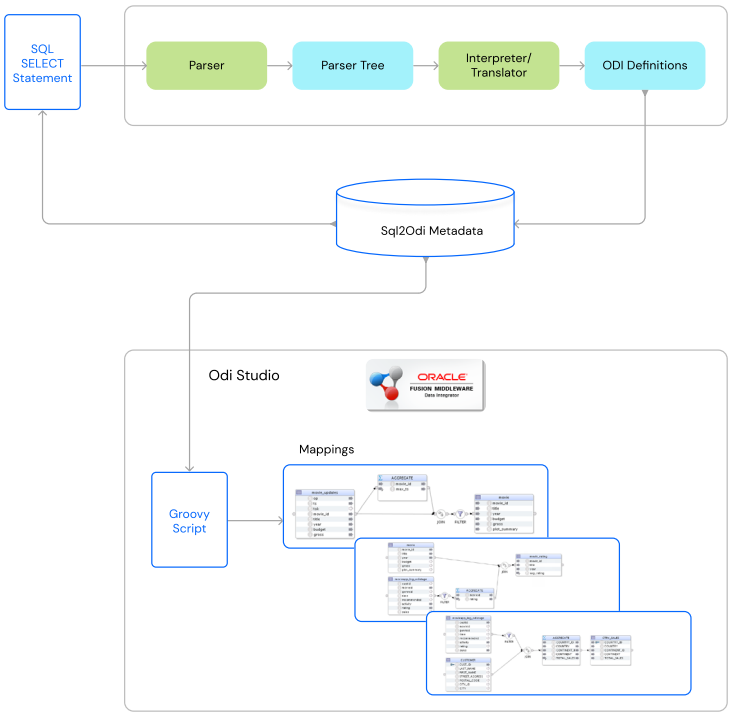


10.1. SQLtoODI

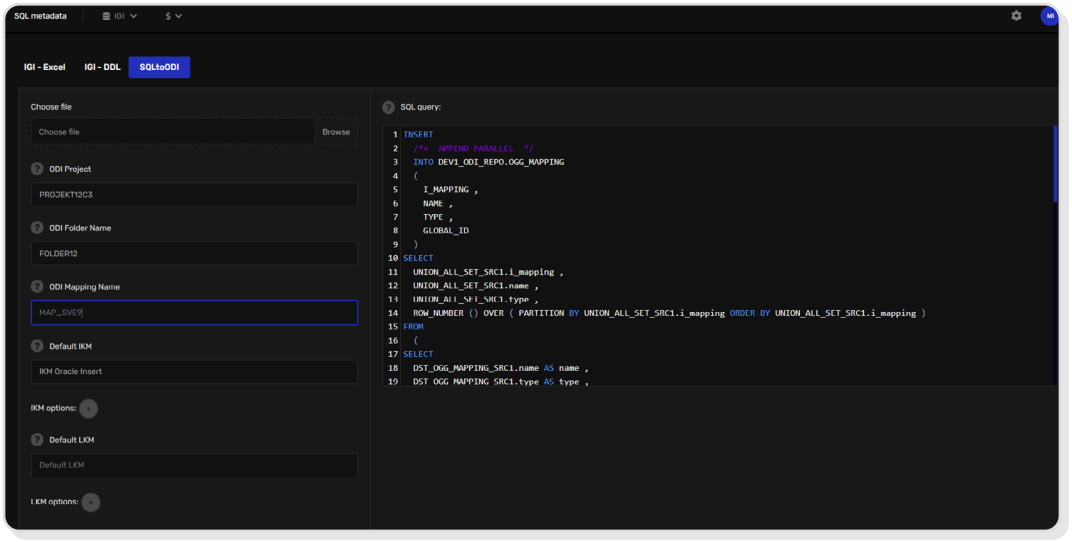
In DWH systems, there is a lot of work involved in creating new and editing existing ODI mappings (transcribing SQL, PL/SQL to ODI mappings). It's hard and time-consuming work for developers. This job requires high precision while adhering to development standards. For large data integration projects, using automation tools is, without a doubt, the best choice. SQLtoODI helps you easily create a large number of ODI objects in a simple and intuitive way with only SQL code. SQLtoODI is a great choice when developing smaller change requests (CRs), and a perfect choice when developing larger CRs, migrating PL/SQL to ODI, migrating OWB to ODI or migrating any form of SQL to ODI. SQLtoODI turns months of manual development into minutes.

We have developed a tool that translates complex SQL SELECT statements into ODI mappings. It consists of three segments:

- an advanced SQL parser developed by Agilos IT, which results in high-speed delivery of customized SQL structures
- parser structure reader and Groovy code generator
- SQLtoODI application, which enables the user to obtain Groovy code based on valid SQL code, which can then be used to generate an ODI mapping in the ODI12c tool (which will generate the same SQL code when executed)



> Figure 10-1 SQLtoODI



> Figure 10-2 SQLtoODI application

This application is very useful in a DWH environment because users often have ready-made SQL queries that need to be transferred to the ODI tool. SQLtoODI can be used in the cloud, but also as a local application. You can read more about the tool in the SQLtoODI whitepaper, which among other things, includes a detailed example of using SQLtoODI.

To summarize, advantages of using SQLtoODI application are:

- **Cost savings** - minimum resources, maximum productivity.
- **Guaranteed quality** - all codes have the same standards and are compatible with the ODI data lineage.
- **Better resource distribution** - projects can be completed with fewer developers and testers.
- **Faster development** - development of ODI mappings that take hours or days can now be created in minutes, or maybe even seconds.

If you're interested in the application, feel free to contact us.

10.2. DataLineage

The DataLineage tool analyzes SQL data structures and/or procedures written in the SQL language, parses them to the lowest level and builds an object model of all metadata elements from them. This object model can later be exported in any form that the user needs, transferred to another tool, or it can be used to create visualizations of data connections, data lineage, or data flow overview, or even scripts for ETL without using classic ETL tools.

The data lineage visualization tool is intended for the analysis of data connections and transformations, in order for business and technical users to clearly determine the flow and connection of data, their interdependence and the impact on subsequent changes.

Data Lineage is intended for generating a metadata model for complex database architectures. Metadata can be generated based on data structures and their links specified in the referential keys, but the real power of the tool is in the generation of models based on complex SQL queries, where by parsing an individual query to the lowest level, an object metadata model and their connection is built for each individual query, which is the basis for the establishment of a data lineage monitoring the life cycle of data.

Monitoring the data lineage, as one of the basic data governance tasks, implies the establishment of a process of understanding the flow of data connections, the processes that connect and transform them and finally consume them. The process of establishing monitoring is based on metadata data models and their connection.

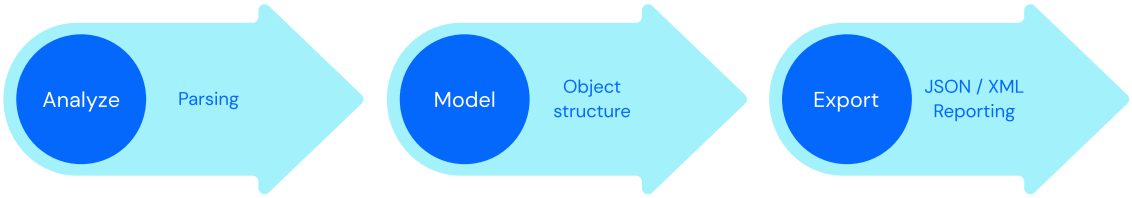
Managing and understanding of metadata structures is the basis of building a system for tracking and visualizing data lineage.

Key features of DataLineage tool:

- Interactive graphic representation of structured metadata
- Advanced user experience based on the needs of actual users, designed by experts with over 10 years of experience working with data warehouse systems
- Effective overview and information retrieval about data flow inside data warehouse
- Automatically generating documentation and advanced expert analysis (which directly frees human resources) such as functional specification (document explaining in detail the effect of future changes to the system).
- Approximating the duration of each implementation
- Generating implementation scripts

- Suggesting advanced expert methods of saving memory capacity using duplicate data detection and unused data detection. This can reduce the need for disk capacity, memory and energy use.

DataLineage makes it possible to define the source of the data structure on the basis of which the metadata model is built. Although the existing structure of data tables in the database can be defined as a source (in this case, the metadata model can be created with basic tools), the real power of this tool is visible when complex procedures or SQL scripts connecting multiple data sources with multiple retrieval criteria are set as the source.



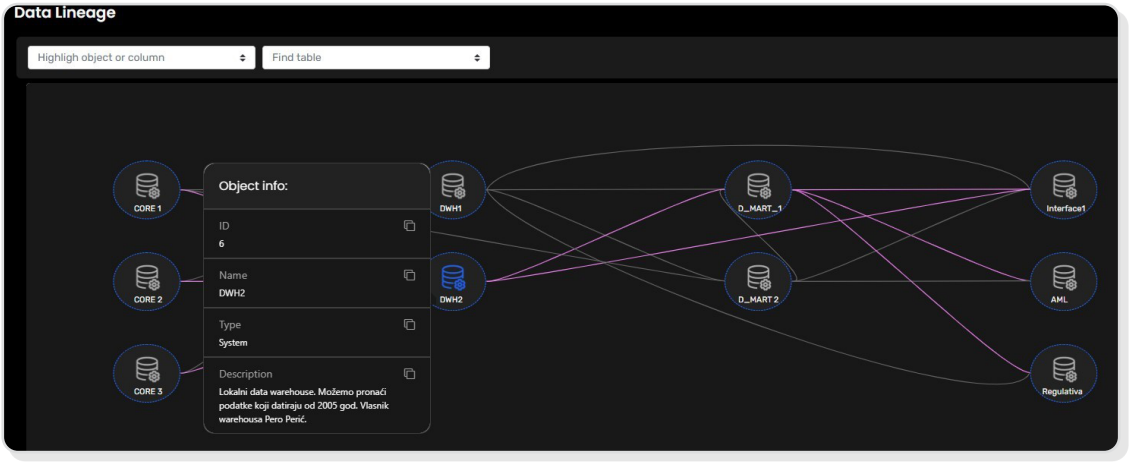
> Figure 10-3 Data Lineage

The first step is parsing of the SQL query, which begins with the separation into basic parts (most often SELECT, FROM, WHERE, etc.), then each of those parts is parsed to the lowest level.

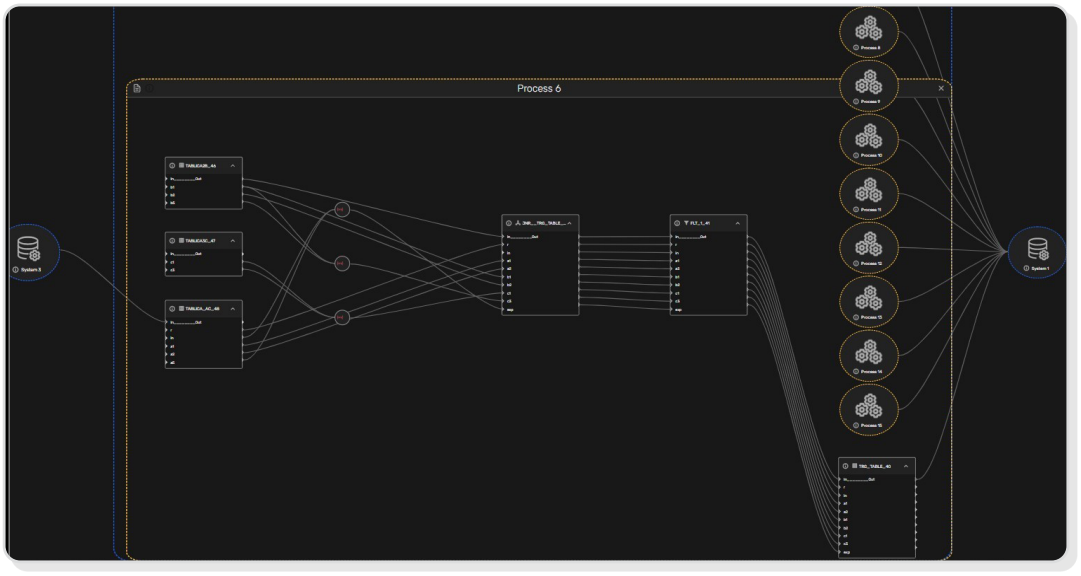
From the parsed data, the application builds a hierarchical object model with all structures and their metadata descriptors and their links.

The final step in implementing the solution is defining the desired structure, format and way of exporting these metadata structures. There are more options, from simple JSON or XML files to creating several types of reports and data lineage visualizations.

DataLineage visualization tool enables visualization of all levels of data lineage.



> Figure 10-4 DataLineage detailed system view



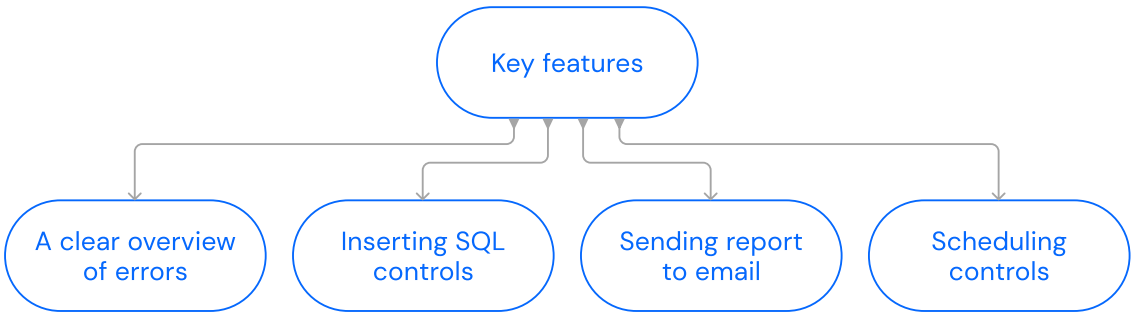
> Figure 10-5 DataLineage detailed ETL(ODI/OWB) view

If you're interested in finding out more about this tool, check out the DataLineage whitepaper or contact us.

10.3. DataQuality

DataQuality is a module of the AgileQuery application and part of the transformation process. The application offers the possibility for the user to quickly and easily recognize data which is incorrect, incomplete or non-existent. Then the user will be able to correct the data so that it could arrive at the final destination completely ready for use.

Maintaining a high level of data quality allows organizations to reduce the cost of identifying and fixing bad data in their systems. In this way, they avoid fines or other additional costs, save on developers who would have to take care of data quality, and maintain a good reputation among their clients.



> Figure 10-6 Key features of the DataQuality tool

The DataQuality application greatly facilitates the work of users, and it also brings financial benefits. It quickly detects problems in the data, and the user receives an overview report directly to the e-mail address, after which the data analyst can consider alternatives for eliminating the cause of the problem, introducing preventive techniques and/or taking some corrective actions. In addition, this tool gives data management teams more time to focus on potentially more productive tasks instead of cleaning data.

> Figure 10-7 DataQuality start screen

If you want to find out more about our DataQuality tool, take a look at DataQuality whitepaper or contact us for more information.

OFFICES

ZAGREB
Oreškovićeva 6 H
10 000 Zagreb, Croatia

SPLIT
Put Žnjana 18 B
21 000 Split, Croatia

CONTACT

T. +385 98 927 1623
info@agilosit.com
https://www.agilos-it.com