The UCSC Data Warehouse

A Cookie Cutter Approach to Data Mart and ETL Development

Alf Brunstrom
Data Warehouse Architect
UCSC ITS / Planning and Budget
abrunstr@ucsc.edu
UCCSC 2016

Agenda

- The UCSC Data Warehouse
- Data Warehouses
- Cookie Cutters
- UCSC Approach
- Real World Examples
- Q&A

The UCSC Data Warehouse

- Managed by Planning & Budget in collaboration with ITS
  - P&B: DWH Manager (Kimberly Register), 1 Business Analyst, 1 User Support Analyst
  - ITS : 3 Programmers
- 20+ years, 1200+ defined users.
- 30+ source systems
- 30+ subject areas
- Technologies
  - SAP Business Objects
  - Oracle 12c on Solaris
  - Toad
  - Git/Bitbucket

Data Warehouses

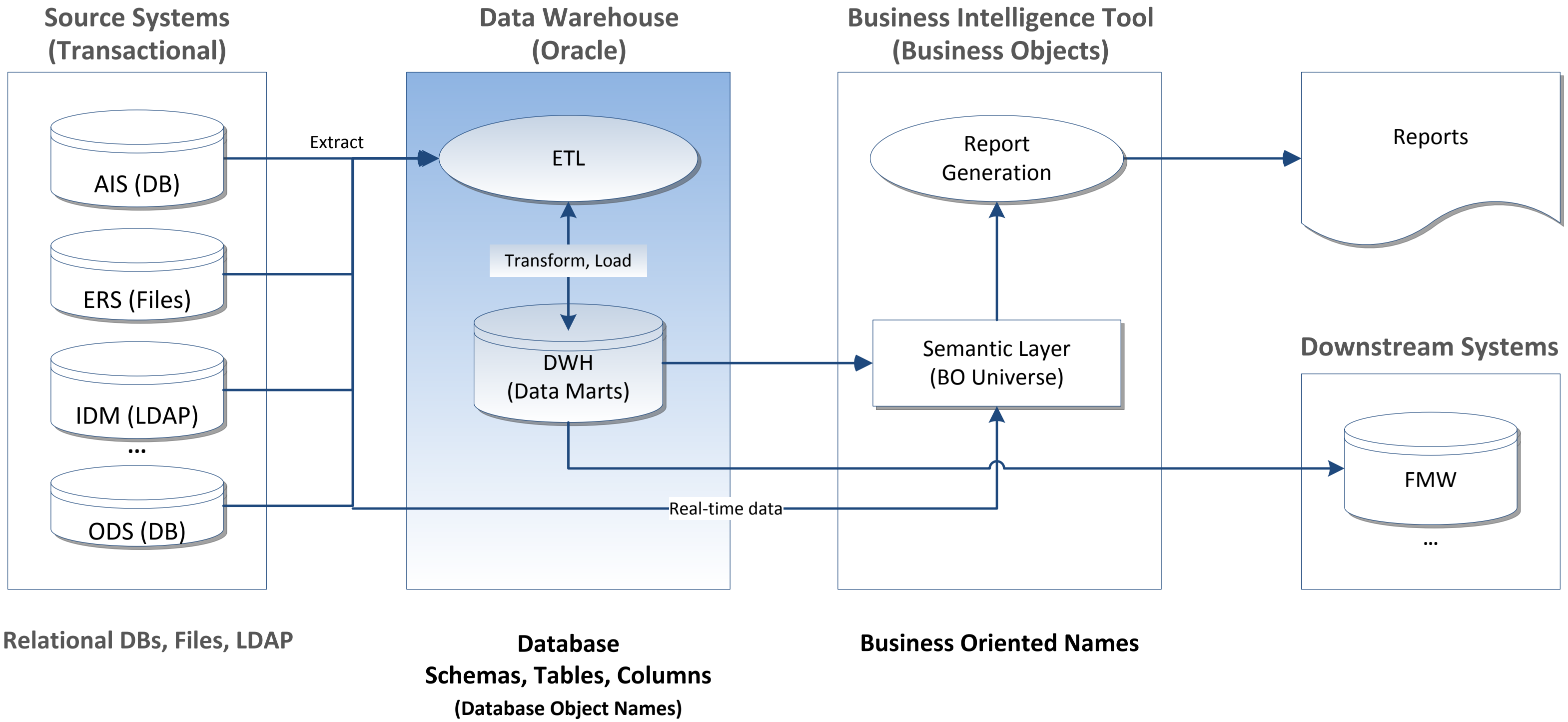
Terminology

- **Data Warehouse**
  - A copy of transaction data specifically structured for query and analysis (Ralph Kimball)
  - A subject oriented, nonvolatile, integrated, time variant collection of data in support of management's decisions (Bill Inmon)

- **Data Mart**
  - A collection of data from a specific subject area (line of business) such as Facilities, Finance, Payroll, etc. A subset of a data warehouse.

- **ETL** (Extract, Transform and Load)
  - A process for loading data from source systems into a data warehouse

Data Warehouse Context Diagram



Source Systems (Transactional)

AIS (DB)

ERS (Files)

IDM (LDAP)

...

ODS (DB)

Extract

Data Warehouse (Oracle)

ETL

Transform, Load

DWH (Data Marts)

Real-time data

Business Intelligence Tool (Business Objects)

Report Generation

Semantic Layer (BO Universe)

Reports

Downstream Systems

FMW

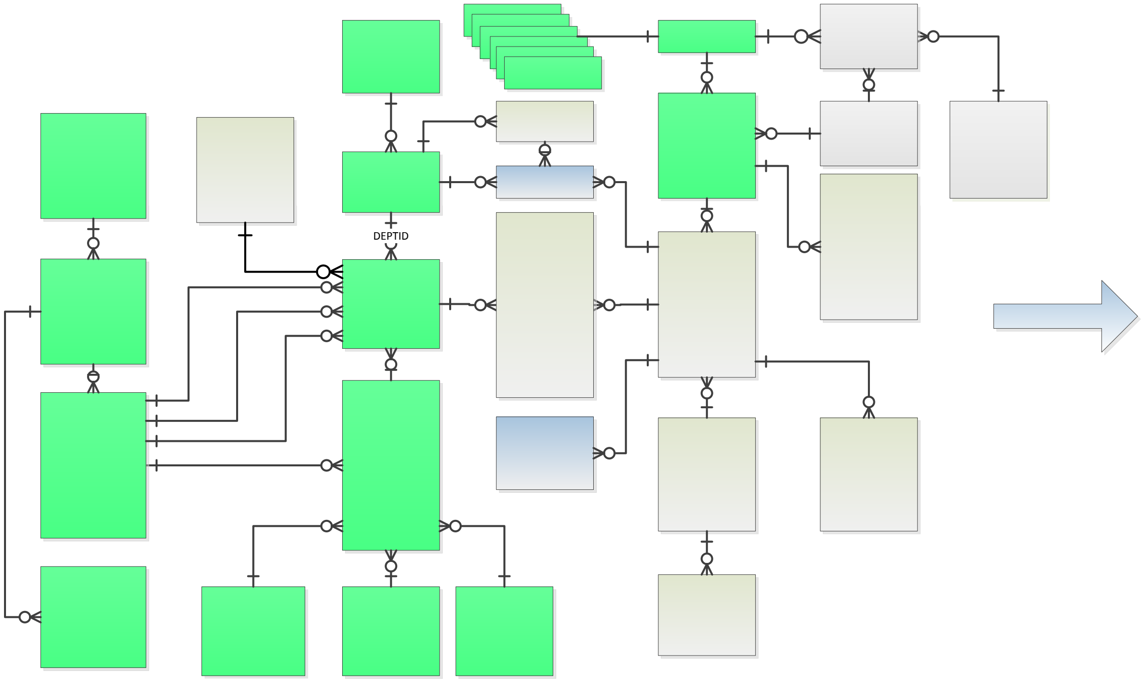
...

Relational DBs, Files, LDAP

Database
Schemas, Tables, Columns
(Database Object Names)

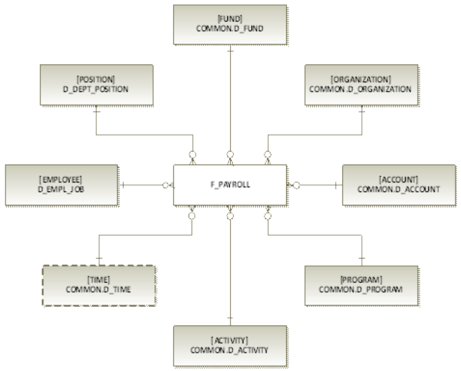
Business Oriented Names

Source System to Data Warehouse Data Structures



Source System

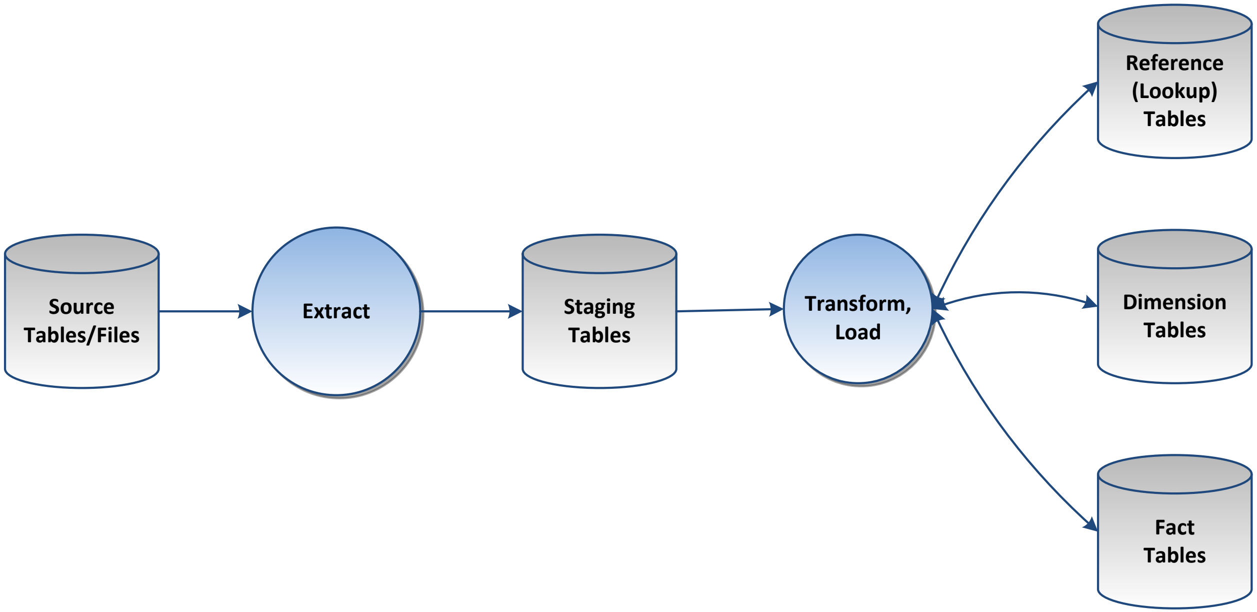
(Normalized Relational)



Data Warehouse

(Dimensional and/or Relational)

Loading the Data Warehouse – Extract, Transform and Load (ETL)



Data Mart Development Tasks

- Analyze requirements, understand the source system data model
- Identify data sources (Tables/Files, columns/fields, keys)
- Specify transformations/calculations
- Design the data model for the data mart
- Design and create tables
    - Types of tables (Staging, reference, dimension, fact and work tables)
    - History / No history
    - Table names
    - Column names, data types, sizes
    - Primary keys
- Define source to target table column mappings
- Design and implement ETL processes to load the data warehouse
- Document
- Test
- Deploy

Example: Creating Table with Primary Key Using DDL

```
CREATE TABLE s_adm_ugrd_release
           (
            student_id                VARCHAR2(11),
            academic_career           VARCHAR2(4),
            career_number             INTEGER,
            admissions_appl_number    VARCHAR2(6),
            appl_program_number       INTEGER,
            effective_date            DATE,
            effective_sequence        INTEGER,
            acadademic_program        VARCHAR2(5),
            program_status            VARCHAR2(4),
            program_action            VARCHAR2(4),
            program_reason            VARCHAR2(4),
            admit_term                VARCHAR2(4),
            admit_type                VARCHAR2(3),
             ...
            etl_job_id                INTEGER,
            etl_job_start             TIMESTAMP(6)
           );

ALTER TABLE s_adm_ugrd_release
  ADD CONSTRAINT s_adm_ugrd_release_pk
           PRIMARY KEY (student_id, academic_career, career_number, admissions_appl_number,
                        appl_program_nbr, effective_date, effective_sequence);
```

- One line of code (LOC) per column (100 columns, 100+ lines of code)
- Can have tens and hundreds of tables in a data mart. Lots of code.

ETL Development Tasks

- Design and implement ETL processes
  - Data extracts from source to staging tables
  - Data conversions
  - Custom transformations (denormalization, aggregation, hierarchy building)
  - Load reference, dimension and fact tables
  - History maintenance (SCD-2)
  - Generate surrogate key values for dimensions
  - Lookup dimension key values when loading fact tables
  - Handle exceptions
  - Manage indexes
  - Logging and Notification (Job id, start time, status, …)
- Test and validate
- Document (Database structure, mappings, data lineage, runbook …)

Example: Loading Table Using DML

```
INSERT INTO s_adm_ugrd_release
           (
            student_id,
            academic_career,
            career_number,
            admissions_appl_number,
            appl_program_number,
            effective_date,
            effective_sequence,
            academic_program,
              ...
           )
    SELECT emplid,
           acad_career,
           stdnt_car_nbr,
           adm_appl_nbr,
           appl_prog_nbr,
           effdt,
           effseq,
           NVL(acad_prog,’-’),
            …
     WHERE <filter>
      FROM dwh_source.cs_adm_ugrd_release@studentdata.ucsc.edu;
```

- Two+ lines of code per column for staging table. More for other types of tables.
- Must map columns in source tables to target tables correctly
- Can have tens or hundreds of tables in a data mart. Lots of code to write from scratch.

- So, without tools there will be lots of coding from scratch, e.g.
  - Column-level DDL scripts to create staging, dimension, fact and other tables and database objects
  - Column-level procedures to extract, transform and load data
  - 25,000+ LOC to process 100 tables not unheard of
- Is there an alternative to buying an expensive commercial ETL tool?
- Especially when funding for such a tool might not be available?

Cookie Cutters

- Cookie Cutter (Noun) (Wikipedia)
  - “A **tool** to cut out cookie dough in a particular shape”
  - “Used where **simplicity** and **uniformity** are required”

- Cookie-Cutter (Adjective) (Merriam-Webster)
  - “**Very similar to other things of the same kind** : not original or different”
  - “Marked by lack of originality or distinction”

- Simplicity and uniformity are desirable properties for data marts
- What if we had “cookie cutters” for developing data marts & ETL?

Cookie Cutters for Data Marts

- What would cookies for data marts look like?
  - Uniformly structured data marts (Schema, directory, source code repository)
  - Uniformly named and structured database objects (tables, constraints, indexes, sequences)
  - Uniformly structured download and ETL jobs

- What types of cookie cutters would we need?
  - Standardized development method
  - Standardized architecture
  - Object creation tools
  - Job templates (ETL and download jobs)
  - ETL runtime services

- What benefits would we get?
  - Common look and feel across data marts (Know one, know all)
  - Faster implementation, easier maintenance, higher quality.
  - Developers can focus on more challenging and interesting tasks

UCSC Approach

A Few Observations and Decisions

- We are loading data into a data warehouse for current and future use.
- Initial reporting requirements are just a starting point. There will be more.
- When extracting data from a table extract all columns, not only the columns you currently need
- User and developer time is expensive.
- Disk space is cheap.
- Don’t spend time renaming columns unless absolutely necessary
  (Reports are developed against the BI semantic layer not physical tables and columns)
- From a high level, data marts are highly similar (Staging, dimension, fact, reference tables, …)
- From a high level, ETL jobs are highly similar (Extract, transform, load, log, …)
- Cookie cutters are great for rapidly creating highly similar cookies
- Software tools are great for rapidly creating highly similar data and software objects

Timeline UCSC Approach

- Late 2012, UCSC Data Mart Project for UCPath
  - No ETL tool available at UCSC DWH. No budget for tool. Coding by hand seemed required.
  - UCPath project undergoing significant changes and growth (120+ tables 2012, 250 tables 2016)
  - Tracking changes time-consuming and error prone if coding by hand.
  - **SETL** “**S**imple **ETL**” architecture and tools proposed and developed to simplify and speed up development
  - SETL originally intended only for the UCPath project. UCSC part of project currently postponed.
- Summer 2013, AIS Operational Reporting
  - Big project. (Admissions, Student Records, Financial Aid, Student Billing) Phased implementation.
  - Implemented using SETL 1.x (Functional enhancements, debugging, tuning)
  - First production data, Nov, 2013. Full production summer 2015.
  - By far the biggest data mart in the UCSC data warehouse. Special requirements.
- 2015 …
  - SETL Version 2.x developed. Cleaner, smaller interface (wrapper for 1.x). Information hiding. Defaults.
  - SETL used for new projects. Converting existing data marts (at least partly) to SETL. In production.
  - SETL = “**S**anta Cruz **ETL**”?

SETL Overview

- Development Method
  - Project setup - Schema, directory, source code repository per data mart
  - Use software tools to create architecture-compliant objects and jobs
  - Design, implement and integrate custom objects and custom code.
  - Document (Runbook)
  - Test and validate.
- Technical Architecture
  - Database - Standardized object names and structures
  - File system - Directory structure standard across data mart
  - ETL jobs - PL/SQL. Template-based. Use SETL ETL runtime services
  - Download jobs - Shell & Python. Template-based. No DB connection to DWH.
  - Interface - Use external tables for loading files
  - Scheduler - Use DBMS_SCHEDULER for both ETL and Download jobs
- Software Tools

SETL Database Object Name Standards

- Table names, including prefix not longer than 22 characters. Prefixes

  D_ - Dimension table

  F_ - Fact tables

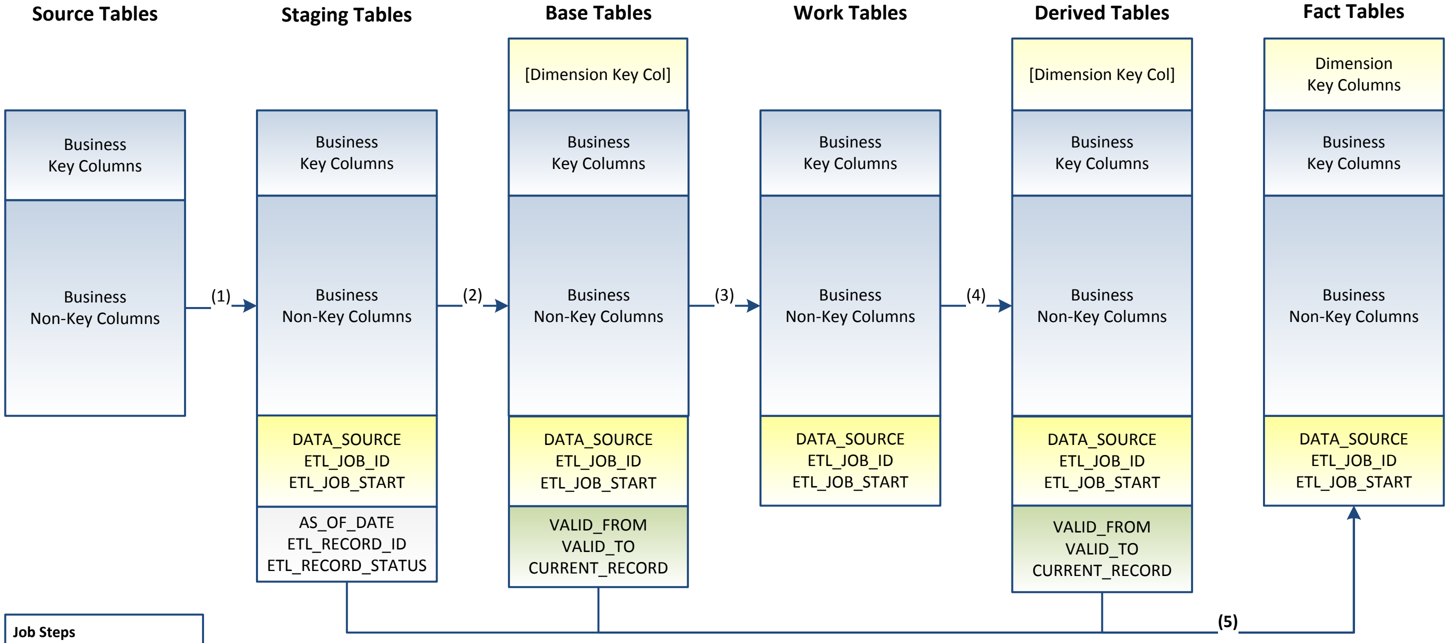
  R_ - Reference tables

  S_ - Staging tables
  X_ - External tables

  W_ - Work tables

- Names of primary keys: <table name>_PK
- Names of indexes: <table name>_IDX<n>
- Names of sequences: <column name>_SEQ
- Names of dimension keys: <dim_table_name>_KEY
- Names, data types and sizes of columns in source tables are preserved in the DWH as far as possible or practical

SETL Table Structures



Source Tables

Business Key Columns

Business Non-Key Columns

(1)

Staging Tables

Business Key Columns

Business Non-Key Columns

DATA_SOURCE
ETL_JOB_ID
ETL_JOB_START

AS_OF_DATE
ETL_RECORD_ID
ETL_RECORD_STATUS

(2)

Base Tables

[Dimension Key Col]

Business Key Columns

Business Non-Key Columns

DATA_SOURCE
ETL_JOB_ID
ETL_JOB_START

VALID_FROM
VALID_TO
CURRENT_RECORD

(3)

Work Tables

Business Key Columns

Business Non-Key Columns

DATA_SOURCE
ETL_JOB_ID
ETL_JOB_START

(4)

Derived Tables

[Dimension Key Col]

Business Key Columns

Business Non-Key Columns

DATA_SOURCE
ETL_JOB_ID
ETL_JOB_START

VALID_FROM
VALID_TO
CURRENT_RECORD

(5)

Fact Tables

Dimension Key Columns

Business Key Columns

Business Non-Key Columns

DATA_SOURCE
ETL_JOB_ID
ETL_JOB_START

Job Steps
(0) – Preprocess (Setup)
(1) - Extract
(2) – Load Base Tables
(3) – Transform
(4) – Load Derived Tables
(5) – Load Fact Tables
(6) - Postprocess

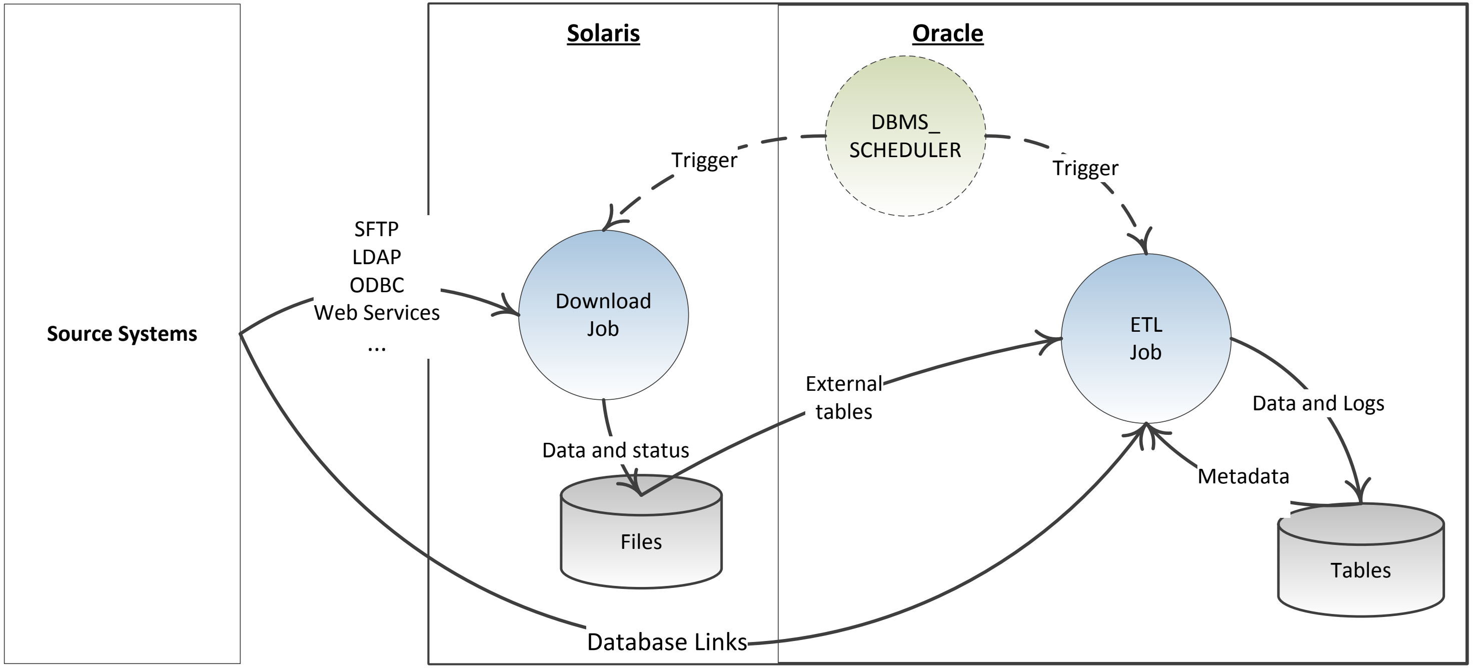
Base Table
A dimension or reference/lookup table loaded directly from a single staging table by a SETL service without requiring any custom code. Typically maintains some form of history.

Derived Table
A dimension or reference/lookup table that cannot be loaded directly from only one staging or base table without undergoing custom transformations via an intermediate work table.

Work Table
Table holding intermediate results of custom transformations, e.g. denormalization, used for loading derived tables.

Alf Brunstrom/UCSC

SETL ETL and Download Jobs

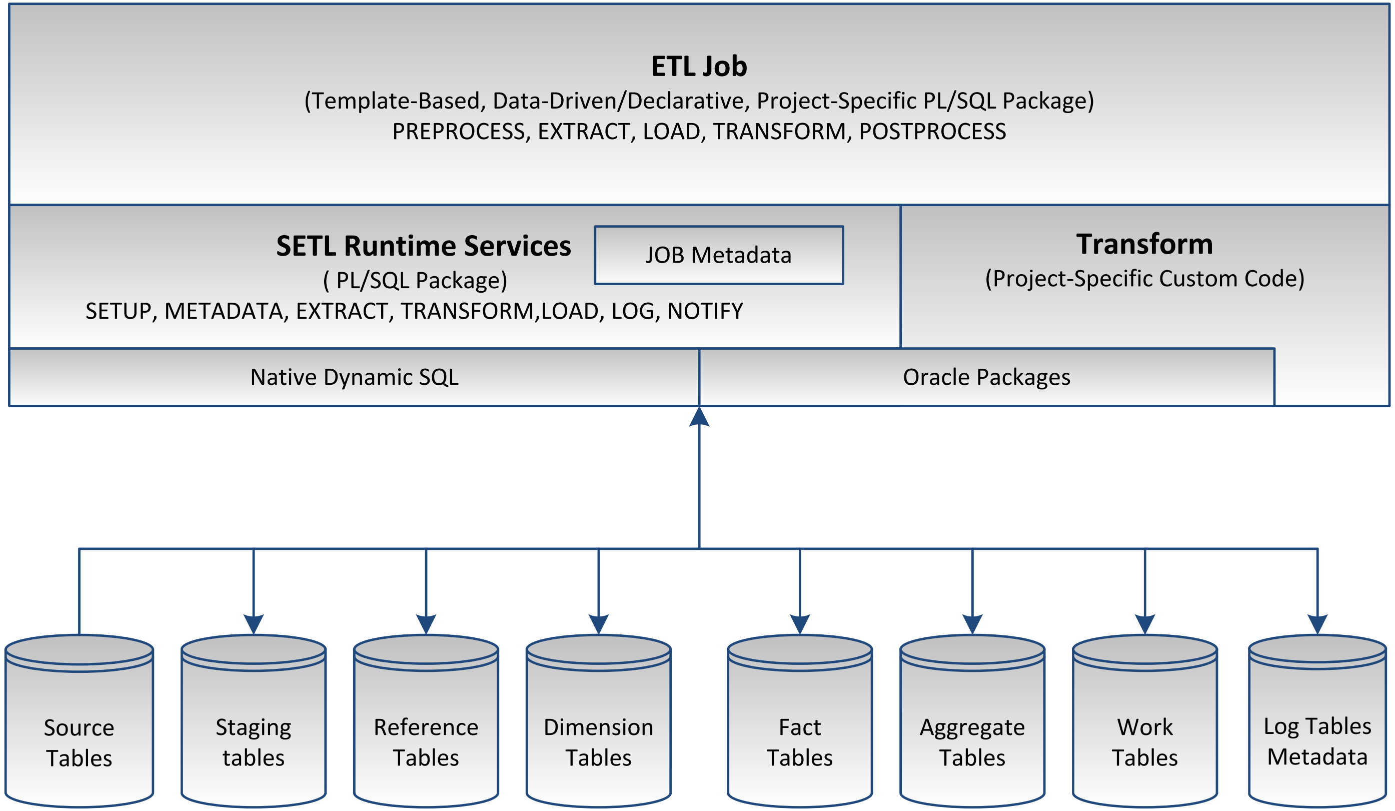


- All jobs (Download and ETL) are executed by Oracle DBMS_SCHEDULER
- Download jobs never connect to the data warehouse
- Download and ETL jobs communicate via files mapped to External Tables

SETL Software Tools

- Database Object Creation Tools
  - Create standardized dimension, fact, reference, staging and other tables
- Directory Structure Creation Tool
  - Create standardized project-specific directory structures on file system
- Job Templates (Download and ETL)
  - Create standardized download and ETL jobs
  - Job, step, target model
  - Data-driven / Declarative (What, not how)
  - High level (Table not column level)
- ETL Runtime Services
  - Common processing tasks (Extract, Load Dimension, Fact, Reference Tables, …)
  - Logging and Notification
  - Metadata export (Data lineage, impact analysis, structural change detection)

ETL Job Structure



SETL Database Object Creation Utilities

Creating Staging Tables

```
/****************************************************************************************
 * STG_TBLS_CREATE.SQL - Create Staging Tables from Source System Tables
 ***************************************************************************************/
BEGIN
    sutl.tbl_cre_ctx_init( 'UCP', 'odsdev.ucsc.edu', 'hcm_ods' );  -- Initialize context (common parameters
        sutl.stg_tbl_create( 's_acct_cd_tbl',      'ps_acct_cd_tbl',     'acct_cd'                          );
        sutl.stg_tbl_create( 's_action_tbl',       'ps_action_tbl',      'action, effdt'                    );
        sutl.stg_tbl_create( 's_actn_reason_tbl',  'ps_actn_reason_tbl', 'action, action_reason, effdt'     );
        sutl.stg_tbl_create( 's_bud_ref_tbl',      'ps_bud_ref_tbl',     'setid, budget_ref, effdt'         );
        sutl.stg_tbl_create( 's_chartfield1_tbl',  'ps_chartfield1_tbl', 'setid, chartfield1, effdt'        );
        sutl.stg_tbl_create( 's_chartfield2_tbl',  'ps_chartfield2_tbl', 'setid, chartfield2, effdt'        );
        sutl.stg_tbl_create( 's_chartfield3_tbl',  'ps_chartfield3_tbl', 'setid, chartfield3, effdt'        );
        sutl.stg_tbl_create( 's_checklist_tbl',    'ps_checklist_tbl',   'checklist_cd, effdt'              );
        sutl.stg_tbl_create( 's_class_cf_tbl',     'ps_class_cf_tbl',    'setid, class_fld, effdt'          );
        sutl.stg_tbl_create( 's_fund_tbl',         'ps_fund_tbl',        'setid, fund_code, effdt'          );
        sutl.stg_tbl_create( 's_addresses',        'ps_addresses',       'emplid, address_type, effdt'      );
        sutl.stg_tbl_create( 's_names',            'ps_names',           'emplid, name_type, effdt'         );
        sutl.stg_tbl_create( 's_person',           'ps_person',          'emplid'                           );
        sutl.stg_tbl_create( 's_visa_permit_tbl',  'ps_visa_permit_tbl', 'country, visa_permit_type, effdt‘ );
END;
/
```

- Tool: SUTL.STG_TBL_CREATE
- A “context” is a set of common parameters referenced by subsequent procedure calls
- Creates table, columns, primary key and records metadata
- One line of code per staging table instead of tens or hundreds
- Rapid table creation. No column name, data type or size errors

Creating Dimension and Reference Tables

```
/****************************************************************************************
 * DIM_REF_TBLS_CREATE.SQL - Create Dimension and Reference Tables From Staging Tables
 ***************************************************************************************/
BEGIN
    -- Create dimension tables
    sutl.tbl_cre_ctx_init( ‘FIS' );
        sutl.dim_tbl_create( 'd_account',      's_fis_v_fzvcact_com' );
        sutl.dim_tbl_create( 'd_activity',     's_fis_v_fzvcatv_com' );
        sutl.dim_tbl_create( 'd_fund',         's_fis_v_fzvcfnd_com' );
        sutl.dim_tbl_create( 'd_organization', 's_fis_v_fzvcorg_com' );
        sutl.dim_tbl_create( 'd_program',      's_fis_v_fzvcprg_com' );

    -- Create reference tables
    sutl.tbl_cre_ctx_init( 'UCP‘ );
        sutl.ref_tbl_create( 'r_action_tbl',      's_action_tbl'      );
        sutl.ref_tbl_create( 'r_actn_reason_tbl', 's_actn_reason_tbl' );
        sutl.ref_tbl_create( 'r_bud_ref_tbl' ,    's_bud_ref_tbl'     );
        sutl.ref_tbl_create( 'r_chartfield1_tbl', 's_chartfield1_tbl' );
        sutl.ref_tbl_create( 'r_chartfield2_tbl', 's_chartfield2_tbl' );
        sutl.ref_tbl_create( 'r_chartfield3_tbl', 's_chartfield3_tbl' );
END;
```

- Tools: SUTL.DIM_TBL_CREATE and SUTL.REF_TBL_CREATE
- Creates tables, columns, PK, indexes, dimension key and metadata
- Columns for SCD-2 history processing (VALID_FROM, VALID_TO, CURRENT_RECORD)
- One line of code per staging table instead of tens or hundreds
- Rapid table creation. No column name, data type or size errors

Creating Fact Tables

```
/****************************************************************************************
 * FACT_TBLS_CREATE.SQL - Example - Create Fact Tables
 ****************************************************************************************/
BEGIN
    sutl.tbl_cre_ctx_init( 'UCP' );

    sutl.fact_tbl_create( 'f_dept_budget',     's_dept_budget',    '[position],[employee]' );
    sutl.fact_tbl_create( 'f_empl_job_action', 's_job',            '[employee],[position],[action]' );
    sutl.fact_tbl_create( ‘f_uc_ll_empl_dtl',  's_uc_ll_empl_dtl', '[position],[employee],[action],[salary_plan]' );
    sutl.fact_tbl_create( 'f_uc_ll_sal_dtl',   's_uc_ll_sal_dtl',
                           '[position],[employee],[fund],[account],[organization],[program],[activity]’ );
END;
```

- Tool: SUTL.FACT_TBL_CREATE
- Dimension names are given within square brackets, e.g. [FUND]
- Referenced dimensions must have been registered before the fact table is created
- Creates tables, columns, primary key, indexes and metadata compatible with Runtime Services
- One line of per fact table instead of tens or hundreds
- Rapid creation. No column name, data type or size errors

SETL-Based ETL Jobs

- SETL Job Model

Job	- Named executable
Step	- Named processing step within a job, e.g. Preprocess, Extract, Transform, …
Target	- Database table being loaded. Each step loads 0 or more targets

- SETL-Based ETL Job

  Template-based PL/SQL package

  Data driven/Declarative (What, not how!)

  Modular, very compact code (Table level)

  Utilizes SETL Runtime Services

  RUN procedure

  SETUP Step (Procedure)

  Custom transformations implemented in the same or a separate PL/SQL package

Run Procedure

```
/***********************************************************************************
 * RUN - Run the HR_DAILY_LOAD ETL process
 ***********************************************************************************/
PROCEDURE run( p_as_of_date TIMESTAMP DEFAULT NULL, p_export_metadata VARCHAR2 DEFAULT setl.c_no ) IS
    export_completed EXCEPTION;
    PRAGMA EXCEPTION_INIT( export_completed, -20100 );
BEGIN
    setl.job_init(p_job_name                => 'hr_daily_load',
                  p_job_description         => 'HR Daily ETL job',
                  p_program_unit            => 'hr_daily_load_ds1.run',
                  p_as_of_date              => job_as_of_date( p_as_of_date ),
                  p_notify_list             => 'hr',
                  p_export_metadata         => p_export_metadata);
    setl.job_start;
    preprocess;
    setl.extract;
    setl.base_tbl_load;
    setl.transform;
    setl.derived_tbl_load;
    setl.fact_tbl_load;
    setl.postprocess;
    setl.job_completed;
    COMMIT;
EXCEPTION
    WHEN export_completed THEN
        setl.job_completed;
        COMMIT;
    WHEN OTHERS THEN
        ROLLBACK;
        setl.job_failed( SQLERRM );
END run;
```

Job Steps

```
setl.job_init( … );
setl.job_start;

preprocess;                 -- Perform job preprocessing/setup
setl.extract;               -- Perform extracts
setl.base_tbl_load;         -- Load base tables (Dim & Ref)
setl.transform;             -- Perform custom transformations
setl.derived_tbl_load;      -- Load derived tables (Dim & Ref)
setl.fact_tbl_load;         -- Load fact tables
setl.postprocess;           -- Perform job post-processing

setl.job_completed;
```

Preprocess Step

```
PROCEDURE preprocess IS
    export_completed EXCEPTION;
    PRAGMA EXCEPTION_INIT( export_completed, -20100 );
BEGIN
    setl.step_start(v_step_name    => ‘PREPROCESS’,
                    v_program_unit => ‘hr_daily_load.preprocess’)
    setup_extracts;
    setup_base_tbl_loads;
    setup_transforms;
    setup_derived_tbl_loads;
    setup_fact_loads;
    setl.job_metadata_export;
    setl.step_completed;
EXCEPTION
    WHEN export_completed THEN
        setl.step_completed;
        RAISE;
    WHEN OTHERS THEN
        setl.step_failed( SQLERRM );
END preprocess;
```

Configuring Extracts

```
/************************************************************************************
 * SETUP_EXTRACTS - Configure Extracts
 ************************************************************************************/
PROCEDURE setup_extracts IS
BEGIN
    -- Set up extracts from UCPath via ODS
    setl.extract_ctx_init('UCP', 'odsdev.ucsc.edu', 'hcm_ods’);
    setl.extract_add('s_action_tbl',      'ps_action_tbl'                            );
    setl.extract_add('s_actn_reason_tbl', 'ps_actn_reason_tbl'                       );
    setl.extract_add('s_addresses',       'ps_addresses'                             );
    setl.extract_add('s_bas_group_tbl',   'ps_bas_group_tbl'                         );
    setl.extract_add('s_bus_unit_tbl_hr', 'ps_bus_unit_tbl_hr'                       );
    setl.extract_add('s_chartfield1_tbl', 'ps_chartfield1_tbl', ' setid=''SCFIN'' ' );
    setl.extract_add('s_chartfield2_tbl', 'ps_chartfield2_tbl', ' setid=''SCFIN'' ' );
    setl.extract_add('s_chartfield3_tbl', 'ps_chartfield3_tbl', ' setid=''SCFIN'' ' );
    setl.extract_add('s_checklist_tbl',   'ps_checklist_tbl'                         );
END setup_extracts;
```

- One line of code per table, not tens or hundreds
- Automatic column to column mapping.
- Automatic logging at table level (Start time, End time, Row count, Status) .
- Full or incremental. Optional deduplication, standard conversions.
- Rapid, error-free implementation.

Configuring Base Table Loads

```
/************************************************************************************
 * SETUP_BASE_TBL_LOADS - Configure base table loads
 ************************************************************************************/
PROCEDURE setup_base_loads IS
BEGIN
    setl.base_tbl_load_ctx_init( 'UCP‘, p_processing_type => ‘scd2’ );
        setl.base_tbl_load_add( 'ref', 'r_acct_cd_tbl',     's_acct_cd_tbl'     );
        setl.base_tbl_load_add( 'ref', 'r_action_tbl',      's_action_tbl'      );
        setl.base_tbl_load_add( 'ref', 'r_actn_reason_tbl', 's_actn_reason_tbl' );
        setl.base_tbl_load_add( 'ref', 'r_addresses',       's_addresses'       );
        setl.base_tbl_load_add( 'ref', 'r_bas_group_tbl',   's_bas_group_tbl'   );
        setl.base_tbl_load_add( 'ref', 'r_bud_ref_tbl',     's_bud_ref_tbl'     );
        setl.base_tbl_load_add( 'ref', 'r_bus_unit_tbl_hr', 's_bus_unit_tbl_hr' );
END setup_base_tbl_loads;
```

- A Base Table is a dimension or reference table that can be loaded directly from a single staging table without any custom transformations.
- One line of code per table. Automatic column mapping. Automatic logging.
- Rapid, error free implementation.

Configuring/Registering Custom Transformations

```
/*************************************************************************************
 * SETUP_TRANSFORMS – Configure/Register Custom Transformations
 *************************************************************************************/
PROCEDURE setup_transforms IS
BEGIN
    setl.transform_ctx_init('hr_transform.load_w_action_reason', 'UCP' );
        setl.transform_add('w_action_reason', 'r_action_tbl'      );
        setl.transform_add('w_action_reason', 'r_actn_reason_tbl' );

    setl.transform_ctx_init('hr_transform.load_w_sal_plan_grade_step', 'UCP' );
        setl.transform_add( 'w_sal_plan_grade_step', 'r_sal_plan_tbl'  );
        setl.transform_add( 'w_sal_plan_grade_step', 'r_sal_grade_tbl' );
        setl.transform_add( 'w_sal_plan_grade_step', 'r_sal_step_tbl'  );
END setup_transforms;
```

- A Custom Transformation involves data from more than one input table and/or complex custom computations
- Custom transformations are implemented in PL/SQL. One procedure per target table.
- Every custom transformation procedure must use the same parameter list

Configuring Derived Table Loads

```
/*************************************************************************************
 * SETUP_DERIVED_TBL_LOADS – Configure Derived Table Loads.
 ************************************************************************************/
PROCEDURE setup_derived_tbl_loads
IS
BEGIN
    setl.derived_tbl_load_ctx_init( 'UCP' );
    setl.derived_tbl_load_add('dim', 'd_action_reason',       'w_action_reason‘      );
    setl.derived_tbl_load_add('dim', 'd_sal_plan_grade_step', 'w_sal_plan_grade_step');
END setup_derived_tbl_loads;
```

- A Derived Table is a table that cannot be loaded directly from a single staging table.
- Examples: Denormalized tables, hierarchical dimension tables or complex report-specific tables
- A derived table is loaded from a complex query or view, or from a work table that has been populated by a custom transformation procedure with data from multiple tables or the results of complex computations.
- Derived tables support the same types of history as base tables

Configuring Fact Table Loads

```
/************************************************************************************
 * SETUP_FACT_TBL_LOADS – Configure Fact Table Loads.
 ************************************************************************************/
PROCEDURE setup_fact_tbl_loads IS
BEGIN
    setl.fact_tbl_load_ctx_init('UCP' );
        setl.fact_tbl_load_add( 'f_empl_job_action', 's_job',
                                '[employee]{};, [position]{};, [action]{};' );

        setl.fact_tbl_load_add( 'f_dept_budget', 's_dept_budget',
                                '[position]{};‘, '[employee]{};‘, 5 );

        setl.fact_tbl_load_add( 'f_uc_ll_sum', 's_uc_ll_sum',
                                '[organization]{deptid_cf};, [fund]{fund_code};, ' ||
                                '[account]{account};, [activity]{activity_id};' );
END setup_fact_tbl_loads;
```

- Names of dimensions are given in square brackets, e.g. [position].
- Fact table key column expressions are given within curly braces, e.g. {deptid_cf}.
- One line of code per table, not hundreds
- Automatic column mapping, dimension key lookup and logging.
- Rapid implementation.

Logging and Notification

- SETL automatically logs the progress and status of each ETL job to log tables in the database as the job executes. Logging is done at the Job, Step and Target levels. No custom code is required.

- SETL notifies members of a job-specific, user-defined mailing list via email or SMS of each job’s completion status. No custom code is required.

Job Control

- **Preconditions** – ETL jobs can be configured to check that one or more conditions are met before full processing starts. If a condition is not met, then the job hibernates for the specified period before checking again. The number of times a job checks preconditions before it times out and raises an exception is configurable.

- **Successors -** ETL jobs can be configured to start other jobs, so called Successors, if they complete successfully.

Export of Job Metadata

Each SETL-based ETL job is driven by metadata. By including only a few lines of extra code in the Preprocess and Run procedures, SETL can export the job metadata to Oracle tables.

The metadata can e.g. be used for

- data lineage analysis
- impact analysis
- detecting changes to source system tables.

For example, the AISOPS (AIS Operational) data mart uses the metadata to detect and report structural differences between referenced tables in the production database and tables in the QA and support databases. This is done on a nightly basis.

SETL Benefits

- Common Look and Feel Across Data Marts (Know one, know all)
  - Database schema, directory structure, source code repository per data mart
  - Object name and structure standards
  - Highly similar ETL and download jobs
- Rapid Development
  - High-level object creation tools (table, directory)
  - Job templates, High-level SETL runtime services → Compact, efficient, very similar code.
  - Simple tools: Toad/SQL Developer, text editor
- Standardized Operations
  - All jobs, ETL and Download, executed by DBMS_SCHEDULER.
  - Single console (Toad/SQL Developer) for controlling and monitoring jobs
  - Logging and notification
- Non-proprietary
  - No expensive software licenses. Deploy as many times as needed.
  - No proprietary data formats or difficult to learn tools. Metadata in text and database format.
  - No lock-in

Real World Examples

payroll_daily_load_ds1_pkg.sql	- Old ETL job converted to SETL
payroll_transform_pkg.sql	- Custom transformations
hr_daily_load_pkg.sql	- SETL ETL job
hr_transform_pkg.sql	- Custom transformations
aisops_etl_summary_rpt_pkg.sql	- Preconditions
hr_stg_tbls_create.sql	- Create staging tables
hr_base_tbls_create.sql	- Create base tables
hr_work_tbls_create.sql	- Create work tables
hr_derived_tbls_create.sql	- Create derived tables
hr_fact_tbls_create.sql	- Create fact tables

Q & A