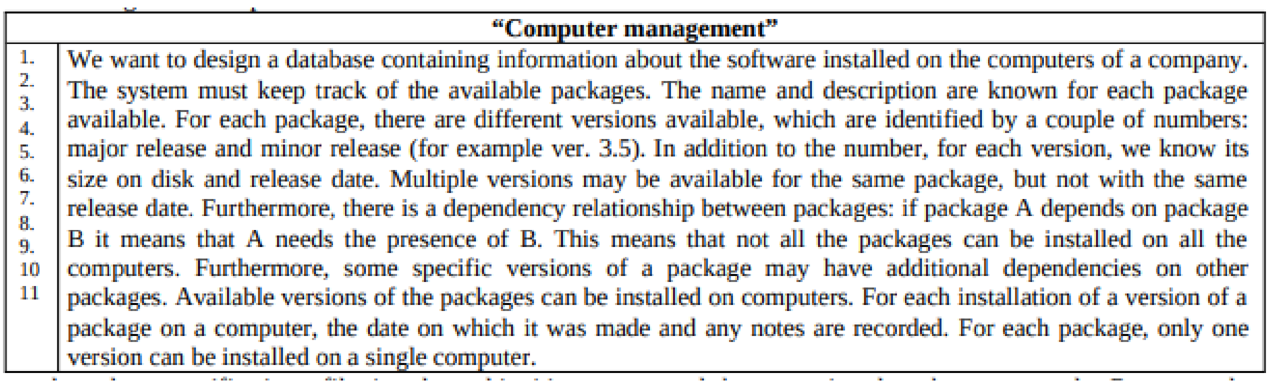
UNIVERSITÀ DEGLI STUDI DI BARI
ALDO MORO

Course of ***Database systems***

PROJECT: COMPUTER MANAGEMENT

Professor:
Prof. Michelangelo Ceci

Student:
Andrea Montemurro



"Computer management"

We want to design a database containing information about the software installed on the computers of a company. The system must keep track of the available packages. The name and description are known for each package available. For each package, there are different versions available, which are identified by a couple of numbers: major release and minor release (for example ver. 3.5). In addition to the number, for each version, we know its size on disk and release date. Multiple versions may be available for the same package, but not with the same release date. Furthermore, there is a dependency relationship between packages: if package A depends on package B it means that A needs the presence of B. This means that not all the packages can be installed on all the computers. Furthermore, some specific versions of a package may have additional dependencies on other packages. Available versions of the packages can be installed on computers. For each installation of a version of a package on a computer, the date on which it was made and any notes are recorded. For each package, only one version can be installed on a single computer.

2) Consider the conceptual scheme defined in the previous exercise. Suppose that the following operations are carried out on this data:

Op1: Verify the possibility of installing a package, knowing the version, on a given PC (160 times a day)
Op2: Install a package (150 times a day)
Op3: Search for computers on which a given package is installed, sorted by version (100 times a week)
Op4: Display of installed packages, including the number of PCs on which they are installed (15 times a day)
Op5: Removal of a computer (10 times a day)

REQUIREMENT ANALYSIS

Choose the right level of abstraction

We don't need to specify other attributes.

Linearize Phrases and break those articulates

There are no excessive complex sentences.

Identify homonyms and synonyms

In the first sentence of the case of study the term "**software**" is used. Then we can see the term "**package**" referring to the same concept. So, we can consider as homonyms and we choose to use only the term **package**.

Reorganizing keyword sentences

We reorganize the specification's sentences grouping them into the most important concepts.

GENERAL SENTENCES: We want to design the database of an application for the computer management of the packages installed on the computers of a company.

PACKAGES SENTENCES: For the packages we will represent using the identifiers package number, package name, package description, package dependency, package size.

VERSIONS SENTENCES: For each version we will use the identifiers version id, release date and disk size and each version are classified into subcategories such as major release and minor release. We can have multiple versions of the same package, but with different release date. Some version can have additional dependencies on other packages.

COMPUTER SENTENCES: For each installation of a version of a package on a computer, the date on which it was made and any notes are recorded. A single computer can only have one installed version per package.

Standardize sentence structure

- For the **package** we are interested in representing number, name, Description, dependency, Size.
- For the **version** we are interested in representing the id, size and release date.
- For the **computer** we are interested in representing the system ID and system type.

Glossary of terms

TERM	DESCRIPTION	SYNONYMS	CONNECTIONS
Package	Packages are installed on computers and are identified by number, name, description, dependencies and size.	Software	Package, Version, Computer
Version	Packages have different versions. Each version has id, release date and size on disk. Only one version of a package can be installed on a specific computer at a time.		Package, Computer
Computer	Computer are machine on which version of packages are installed. We assume each computer have an ID		Version

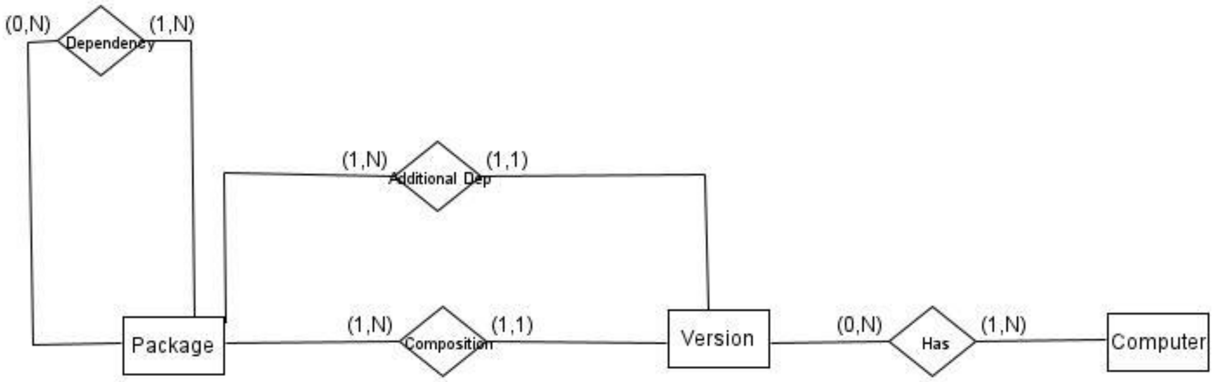
	and a type. (Was not in the specifications)		

CONCEPTUAL DESIGN

The strategy followed in the phase of conceptual modelling is the hybrid strategy: starting from the specifications, we will represent all the information in an initial skeleton using a few abstract concepts.

Then, each entity of the scheme will be refined and the different schemes obtained will be integrated, leading to the final ER scheme, much more detailed than the initial one.

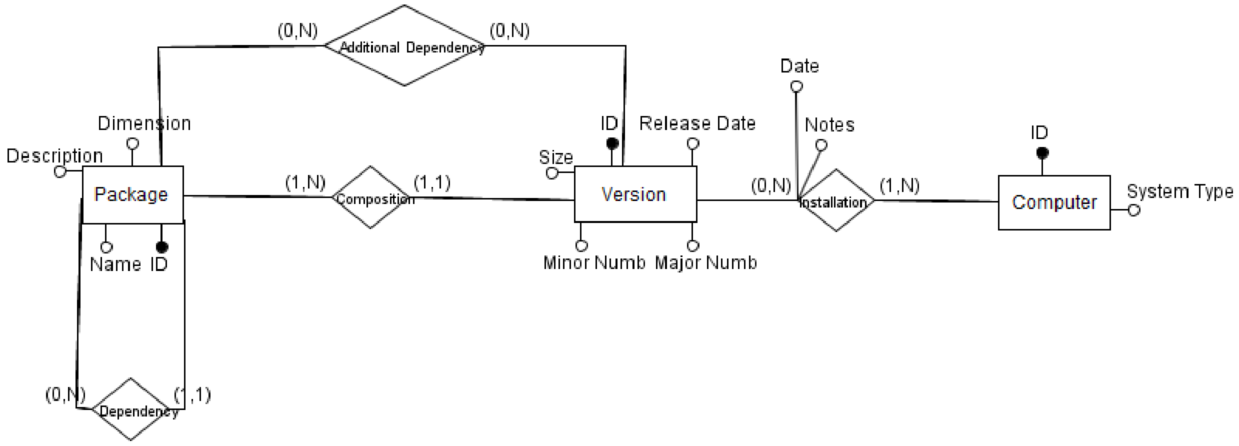
Skeleton Schema



Now, we can refine the schema by adding the attributes for each entity the attributes:

- The **package** must be identified by a Number, have a name, a size, and a description.
  Moreover a package is linked with itself with the relation **Dependency.**
- The **version** is also identified by an ID, size and release date. Moreover for the version the specification “suggest” to use generalization in order to distinguish from minor and major release, but to avoid redundancies we can use just two numerical attributes to identify major release number and minor release number.
  A package has more version, so Version is linked to package by the relation **Composition**.
  Since a version can require other packages, it is linked with package by using the relation **Additional Dependency**.
- A **computer** is identified by an ID, and the system type. On a computer we can install versions, and a version can be installed on more than one computer. So, it is linked by a multi-multi relation.
  We can add am additional entity to solve this kind of relation, adding a table **Installation in the logical phase.**
- An **Installation** is identified by an ID, and has a date, and notes. It is linked to **Computer** and **Version.**

Final ER Schema



Constraints

- A single version of a specific package can be installed on a computer a time. So we have to set something to check such a constraint.
- Multiple versions can have for a specific package, but with different release date. We also have to check this.

LOGICAL DESIGN

Before starting with the implementation of the types and tables for the database, we decide if it is necessary to restructure schema from the previous phase. For completeness we will make the volumes table and the access tables. In the volumes tables we add a column to explain the assumptions taken and to the access tables we add a column to explain the assumption taken when necessary.

Volume table

Concept	Type	Volume	Explanation
Computer	E	600	From Specifications
Version	E	50	Assuming on average 5 major release and for each major 10 minor release
Package	E	50	Assumptions
Dependency	R	2500	Assuming on average 5 dependencies for each package
Composition	R	2500	
Additional Dependency	R	150	Assuming on average 3 additional dep. For each version
Installation	R	3000	

Operation table

Operation	Type	Frequency
Op1: Verify the possibility of installing a package, knowing the version, on a given PC	I	160/day
Op2: Install a package	I	50/day

Op3: Search for computers on which a given package is installed, sorted by version	I	100/week
Op4: Display of installed packages, including the number of PCs on which they are installed	B	15/day
Op5: Removal of a computer	I	10/day

Access Tables

We create the access tables for the operation given in specification, in order to see if the redundancies are useful for the operations:

Op1: Verify the possibility of installing a package, knowing the version, on a given PC.

CONCEPT	CONSTRUCT	ACCESS	TYPE
Package	E	1	R
Composition	R	50	R
Version	E	1	R
Installation	R	50	R
Computer	E	1	R

Op2: Install a package.

CONCEPT	CONSTRUCT	ACCESS	TYPE
Computer	E	1	R
Installation	R	50	R
Version	E	1	R
Composition	R	50	R
Package	E	1	R
Dependency	R	5	R
Additional Dependency	R	3	R
Installation	R	1	W

Op3: Search for computers on which a given package is installed, sorted by version

CONCEPT	CONSTRUCT	ACCESS	TYPE
Package	E	1	R
Composition	R	50	R
Version	E	1	R
Installation	R	50	R
Computer	E	1	R
Installation	R	50	R
Version	E	1	R

Op4: Display of installed packages, including the number of PCs on which they are installed

CONCEPT	CONSTRUCT	ACCESS	TYPE
Package	E	1	R
Composition	R	50	R
Version	E	1	R
Installation	R	50	R
Computer	R	1	R

Op5: Remove a computer

CONCEPT	CONSTRUCT	ACCESS	TYPE
Computer	E	1	W
Installation	R	50	W

We have no redundant attributes nor generalization (we already decide to differentiate minor and major release in Version by using just two numerical attributes).
So the final schema remain the same.

IMPLEMENTATION (Oracle OBJECT RELATIONAL MODEL)

Types and tables definition

We choose to use an attribute as identifier for each entity.
As we can see from the implementation, we need also additional table to model the recursive relation Dependency from each package (PAKS_DEPS_TB) and additional tables to model the n-n relationship Additional Dependency (ADD_DEPENDENCY_TB) and Installation (INSTALLATION_TB).

TYPES	TABLE
create or replace TYPE **PACKAGE_TY** AS OBJECT (package_ID INTEGER, package_name VARCHAR(10), package_description VARCHAR(20), package_size INTEGER)FINAL;	CREATE TABLE **PACKAGE_TB** OF PACKAGE_TY (package_ID PRIMARY KEY, package_name NOT NULL, package_description NOT NULL, package_size NOT NULL);

CREATE OR REPLACE TYPE **PACK_DEPS_TY** AS OBJECT
(
dep_ID INTEGER,
package_ID REF PACKAGE_TY,
package_dep REF PACKAGE_TY
)FINAL;
	CREATE TABLE **PACK_DEPS_TB** OF PACK_DEPS_TY
	(
	dep_ID PRIMARY KEY
	);

CREATE OR REPLACE TYPE **VERSION_TY** AS OBJECT
(
version_ID INTEGER,
major_rel INTEGER,
minor_rel INTEGER,
release_date DATE,
version_size INTEGER,
package_dep REF PACKAGE_TY
)FINAL;
	CREATE TABLE **VERSION_TB** OF VERSION_TY
	(
	version_ID PRIMARY KEY
	);

CREATE OR REPLACE TYPE **ADD_DEPENDENCY_TY** AS OBJECT
(
add_dep_ID INTEGER,
version_dep REF VERSION_TY,
package_dep REF PACKAGE_TY
)FINAL;
	CREATE TABLE **ADD_DEPENDENCY_TB** OF ADD_DEPENDENCY_TY
	(
	add_dep_ID PRIMARY KEY
	);

CREATE TYPE **INSTALLATION_TY** AS OBJECT
(
installation_ID INTEGER,
installation_notes VARCHAR(20),
installation_date DATE,
installation_version REF VERSION_TY,
installation_computer REF COMPUTER_TY
)FINAL;
	CREATE TABLE **INSTALLATION_TB** OF INSTALLATION_TY
	(
	installation_ID PRIMARY KEY
	);

CREATE OR REPLACE TYPE **COMPUTER_TY** AS OBJECT (computer_ID INTEGER, system_description VARCHAR(20))FINAL;	CREATE TABLE **COMPUTER_TB** OF COMPUTER_TY (computer_ID PRIMARY KEY, system_description NOT NULL);

Schema Population

We randomly populate the previously defined schema by using the following procedures.

- **populate_computer_tb** insert 600 rows in the COMPUTER_TB;

```
CREATE OR REPLACE PROCEDURE populate_computer_tb AS
iteration number;
BEGIN
iteration := 1;
LOOP
INSERT INTO COMPUTER_TB VALUES (COMPUTER_TY(iteration, DBMS_RANDOM.STRING('U', 20)));
iteration := iteration + 1;
EXIT WHEN iteration > 600;
END LOOP;
END;
```

- **POPULATE_PACKAGE_TB** insert randomly 50 rows in PACKAGE_TB;

```
CREATE OR REPLACE PROCEDURE POPULATE_PACKAGE_TB AS
iteration number;
BEGIN
iteration := 1;
LOOP
INSERT INTO PACKAGE_TB VALUES (PACKAGE_TY(iteration, DBMS_RANDOM.STRING('U', 10),
DBMS_RANDOM.STRING('U', 20),
(select trunc(dbms_random.value(0, 50),0) from dual)));
iteration := iteration + 1;
EXIT WHEN iteration > 50;
END LOOP;
END;
```

- **POPULATE_PACKAGE_DEPS_TB** insert randomly 250 dependencies for packages by choosing randomly ref from the Package table to package objects;

```
CREATE OR REPLACE PROCEDURE POPULATE_PACKAGE_DEPS_TB AS
iteration number;
BEGIN
iteration := 1;
LOOP
INSERT INTO PACK_DEPS_TB VALUES (iteration,
(SELECT package_ref FROM ( SELECT REF(P) package_ref FROM PACKAGE_TB P ORDER BY dbms_random.value)
WHERE rownum = 1),
(SELECT package_ref FROM ( SELECT REF(P) package_ref FROM PACKAGE_TB P ORDER BY dbms_random.value)
WHERE rownum = 1));
iteration := iteration + 1;
EXIT WHEN iteration > 250;
END LOOP;
END;
```

- **POPULATE_VERSION_TB** insert randomly 2500 versions of packages (5 major releases and 10 major releases for each package in average);

```
CREATE OR REPLACE PROCEDURE POPULATE_VERSION_TB AS
iteration number;
BEGIN
iteration := 1;
LOOP
-- INSERIMENTO versioni scegliendo ref casuali
INSERT INTO VERSION_TB VALUES (VERSION_TY(iteration,
(select trunc(dbms_random.value(0, 5),0) from dual),
(select trunc(dbms_random.value(0, 10),0) from dual),
(SELECT to_date(trunc(dbms_random.value(to_char(DATE '2000-01-01', 'J'), to_char(DATE '2022-12-31', 'J'))), 'J')FROM
DUAL),
(select trunc(dbms_random.value(0, 50),0) from dual),
(SELECT package_ref FROM ( SELECT REF(P) package_ref FROM PACKAGE_TB P ORDER BY dbms_random.value)
WHERE rownum = 1)));
iteration := iteration + 1;
EXIT WHEN iteration > 2500;
END LOOP;
END;
```

- **POPULATE_ADDITIONAL_DEPS_TB** insert the additional dependencies for the versions by choosing randomly the ref to the package objects;

```
CREATE OR REPLACE PROCEDURE POPULATE_ADDITIONAL_DEPS_TB AS
iteration number;
BEGIN
iteration := 1;
LOOP
-- INSERIMENTO dipendenze scegliendo ref casuali
INSERT INTO add_dependency_tb VALUES (ADD_DEPENDENCY_TY(iteration,
(SELECT version_ref FROM ( SELECT REF(V) version_ref FROM VERSION_TB V ORDER BY dbms_random.value)
WHERE rownum = 1),
(SELECT package_ref FROM ( SELECT REF(P) package_ref FROM PACKAGE_TB P ORDER BY dbms_random.value)
WHERE rownum = 1)));
iteration := iteration + 1;
EXIT WHEN iteration > 150;
-- avgh of 5 dep per pack
END LOOP;
END;
```

- **POPULATE_INSTALLATION_TB** insert the installation object in the table, by choosing randomly the ref to computer objects and version objects.

```
CREATE OR REPLACE PROCEDURE POPULATE_INSTALLATION_TB AS
iteration number;
BEGIN
iteration := 1;
LOOP
-- INSERIMENTO versioni e computer scegliendo ref casuali
INSERT INTO INSTALLATION_TB VALUES (INSTALLATION_TY(iteration,
DBMS_RANDOM.STRING('U', 20),
(SELECT to_date(trunc(dbms_random.value(to_char(DATE '2000-01-01', 'J'), to_char(DATE '2022-12-31', 'J'))), 'J')FROM
DUAL),
(SELECT version_ref FROM ( SELECT REF(V) version_ref FROM VERSION_TB V ORDER BY dbms_random.value)
WHERE rownum = 1),
(SELECT computer_ref FROM ( SELECT REF(C) computer_ref FROM COMPUTER_TB C ORDER BY
dbms_random.value) WHERE rownum = 1)
));

iteration := iteration + 1;
EXIT WHEN iteration > 30000;
-- avgh of 5 major rel and 10 minor per pack
END LOOP;
```

ACTIVE DATABASE

Trigger implementation

We implement triggers to manage the constraint that we previously define which we could not handle when defining the schema of tables and objects.
The previously defined constraint was:

1. A single version of a specific package can be installed on a computer a time. So we have to set something to check such a constraint.
2. Multiple versions can have for a specific package, but with different release date. We also have to check this.

So, the the trigger called **CHECK_SAME_PACKAGE** is defined to manage the constraint nr 1.
Such a trigger before an insert, count how many versions we already have for the same package with same computer and if the number is greater than 0 raise an application error and print a message for the user.

```
CREATE OR REPLACE TRIGGER CHECK_SAME_PACKAGE
BEFORE INSERT ON INSTALLATION_TB
FOR EACH ROW
DECLARE existPackage NUMBER;

BEGIN
-- controllo se il package risulta gi'a installato
SELECT COUNT(*) INTO existPackage FROM INSTALLATION_TB, VERSION_TB, PACKAGE_TB
WHERE deref(installation_computer).computer_ID = deref(:new.installation_computer).computer_ID
AND deref(installation_version).version_ID = VERSION_TB.version_ID
AND deref(package_dep).package_iD = PACKAGE_TB.package_ID;

IF (existPackage>0) THEN
   raise_application_error(-20012, 'You can install a version a time of the same package on the same computer ');
END IF;
END;
```

Similarly, the trigger called **CHECK_SAME_PACKVERS_DATE** checks if there are more version with same release date and in case, raise an application error printing a message for the user.

```
CREATE OR REPLACE TRIGGER CHECK_SAME_PACKVERS_DATE
BEFORE INSERT ON VERSION_TB
FOR EACH ROW
DECLARE existPackage NUMBER;

BEGIN
-- controllo se ci sono gi'a versioni con stessa data dello stesso package
SELECT COUNT(*) INTO existPackage FROM VERSION_TB
WHERE deref(package_dep).package_ID = deref(:new.package_dep).package_ID
AND release_date = :new.release_date;

IF (existPackage>0) THEN
   raise_application_error(-20012, 'Can t have more version with same release date for the same package ');
END IF;
END;
```

Operations implementation

PL/SQL Procedures

1. **CHECK_PACKAGE_INSTALLATION** (OP1) is the procedure that check if it is possible to install a specified versions of a package on a given pc. In input it needs the identifiers of the computer and of the package version.
   A query is performed in order to count how many installations are already done with the same package on that computer, and if there are not already installed versions it prints a massage for the user. In the other case, an application error is raised with an error message.

```
            CREATE OR REPLACE PROCEDURE CHECK_PACKAGE_INSTALLATION (computerID INTEGER, versionID
            INTEGER) AS
            packageID INTEGER;
            alreadyInstalled NUMBER;

            BEGIN
            alreadyInstalled := 0;

            -- retry the package id for the version of interest
            SELECT p.package_ID INTO packageID
            FROM VERSION_TB v, PACKAGE_TB p
            WHERE v.version_ID = versionID
            AND DEREF(v.package_dep).package_ID = p.package_ID;


            -- check if there are other installation on the given pc for the same version' s package
            SELECT COUNT(*) INTO alreadyInstalled
            FROM COMPUTER_TB c, INSTALLATION_TB i, VERSION_TB v
            WHERE deref(i.installation_version).version_id = v.version_ID
            AND deref(v.package_dep).package_ID = packageID
            AND deref(i.installation_computer).computer_id = computerID;

            IF (alreadyInstalled > 0) THEN
               raise_application_error(-20013, 'Another version of the same package is already installed in this computer');
            ELSE
               dbms_output.put_line('You can proceed with the installation of this version on this computer ...');
            END IF;
END;
```

2. **INSTALL_PACKAGE (OP2)**is the procedure that install a package by adding new rows in the installation table. Since a package can require other dependencies, the procedure check for the required dependencies and if are not already installed, an error message is printed for the user.

```
      create or replace PROCEDURE INSTALL_PACKAGE (packageID INTEGER, computerID INTEGER) AS
      packdepid iNTEGER;
      countInstallation INTEGER;
      packdepname VARCHAR(10);
      computerRef ref COMPUTER_TY;
      versRef ref VERSION_TY;
      packageRef ref PACKAGE_TY;
      numDependencies INTEGER;
```

```
-- INSTALLARE PRIMA LE DIPENDENZE DEL PACKAGE
CURSOR depCursor IS (
SELECT dep_ID, deref(p.package_Dep).package_ID FROM PACK_DEPS_TB p
WHERE DEREF(p.package_id).package_id = packageID);


dependency depCursor%rowtype;

BEGIN
packdepid :=0;
-- cHECK HOW MANY DEPENDENCIES ARE REQUIRED
SELECT count(*) INTO numDependencies FROM PACK_DEPS_TB p  WHERE  DEREF(p.package_id).package_id
= packageID;

IF (numDependencies > 0) Then
OPEN depCursor;
LOOP
FETCH depCursor INTO dependency;
EXIT WHEN depCursor%NOTFOUND;
DBMS_OUTPUT.PUT_LINE('DEPENDENCY REQUIRED: ' || dependency.dep_ID);
END LOOP;
CLOSE depCursor;
RAISE_APPLICATION_ERROR(-20000, 'Install the package after the installation of the required dependencies.');
END IF;

--retrieve last package version
SELECT REF(v) INTO versRef
FROM VERSION_TB v
WHERE  DEREF(v.package_dep).package_ID = packageID
ORDER BY v.release_date DESC
FETCH FIRST 1 ROWS ONLY;

-- COUNT installations
SELECT COUNT(*) into countInstallation FROM INSTALLATION_TB;

--retrieve the computer reference
SELECT REF(t) INTO computerRef FROM COMPUTER_TB t WHERE t.computer_ID = computerID;

-- STORE IN INSTALLATION
INSERT INTO INSTALLATION_TB VALUES(
INSTALLATION_TY( (countInstallation+2),
DBMS_RANDOM.STRING('U', 20),
(SELECT to_date(trunc(dbms_random.value(to_char(DATE '2000-01-01', 'J'), to_char(DATE '2022-12-31', 'J'))),
'J')FROM DUAL),
versRef, computerRef));

DBMS_OUTPUT.PUT_LINE('Package succesfully installed');
END;
```

3. **SEARCH_COMPUTER_PACKAGE** (OP3) is the procedure that takes as input the id of a given package, and search for computers on which the given package is installed. The query also orders the result with respect to the release date of installed version of the package on the computer and then the results are printed using a cursor. In case of the given package is not installed on any computer, an error message is printed for the user.

```
create or replace PROCEDURE SEARCH_COMPUTER_PACKAGE(packageID INTEGER) AS
existComputers iNTEGER;

CURSOR computerCursor IS
    SELECT DISTINCT DEREF(i.installation_computer).computer_id AS computerID, v.release_date as reldate
    FROM INSTALLATION_TB i, VERSION_TB v, COMPUTER_TB c
    WHERE DEREF(i.installation_version).version_ID = v.version_id
    AND DEREF(v.package_dep).package_ID = packageID
    AND DEREF(i.installation_computer).computer_ID = c.computer_ID
    ORDER BY v.release_date;

computers computerCursor%rowtype;


BEGIN
    existComputers := 0;
    SELECT COUNT(*) INTO existComputers
    FROM INSTALLATION_TB i, VERSION_TB v, COMPUTER_TB c
```

```
          WHERE DEREF(v.package_dep).package_ID = 1
          AND DEREF(i.installation_computer).computer_ID = c.computer_ID;

          IF (existComputers>0) THEN
            OPEN computerCursor;
            LOOP
            FETCH computerCursor INTO computers;
            EXIT WHEN computerCursor%NOTFOUND;
            DBMS_OUTPUT.PUT_LINE('COMPUTER ID: ' || computers.computerID || ' VERSION RELEASE DATE ' ||
        computers.reldate);
            END LOOP;
            CLOSE computerCursor;
          ELSE
            RAISE_APPLICATION_ERROR(-20000, 'There are no computer with this package installed on.');
          END IF;
END;
```

4. **SHOW_PACK_INSTALLED_COMPUTER** (OP4) is the procedure that print, for each package, the number of computer on which it is installed.

```
        create or replace PROCEDURE SHOW_PACK_INSTALLED_COMPUTER AS
        existpackage INTEGER;

        CURSOR queryCursor IS (

        SELECT DISTINCT p.package_ID as packageID, COUNT (*) AS countComp
        FROM PACKAGE_TB p, VERSION_TB v, INSTALLATION_TB i, COMPUTER_TB c
        WHERE DEREF(v.package_dep).package_ID = p.package_ID
        AND DEREF(i.installation_version).version_ID = v.version_ID
        AND DEREF(i.installation_computer).computer_ID = c.computer_ID
        GROUP BY p.package_ID
        );
        requery queryCursor%rowtype;


        BEGIN

          OPEN queryCursor;
          LOOP
          FETCH queryCursor INTO requery;
          EXIT WHEN queryCursor%notfound;
          DBMS_OUTPUT.PUT_LINE('LIST OF PACKAGE');
          DBMS_OUTPUT.PUT_LINE('ID PACKAGE: ' || requery.packageID || ' PC ON WHICH IS INSTALLED: ' ||
        requery.CountComp);
          DBMS_OUTPUT.PUT_LINE(' ');
          END LOOP;
          CLOSE queryCursor;
END;
```

5. **REMOVE_COMPUTER()** (OP5) Is the procedure that, given as input the id of computer, removes the rows of the given computer from the Computer table and the installation table, because we want to avoid to have dangling reference in such a table (that contains REF to computer table and vesion table).

```
        create or replace PROCEDURE REMOVE_COMPUTER(computerID INTEGER) AS
        existComputer INTEGER;

        BEGIN
          existcomputer := 0;
          SELECT COUNT(*) into existcomputer
          FROM COMPUTER_TB WHERE computer_id = computerID;

          IF (existcomputer>0) THEN
            DELETE FROM INSTALLATION_TB
            WHERE DEREF(installation_computer).computer_ID = computerID;

            DELETE FROM computer_tb
            WHERE computer_id=computer_ID;

            DBMS_OUTPUT.PUT('Computer DELETEDfrom the database  with all its installations');

          ELSE
            RAISE_APPLICATION_ERROR(-20000, 'The computer does not exist, can t remove it.');
          END IF;
```

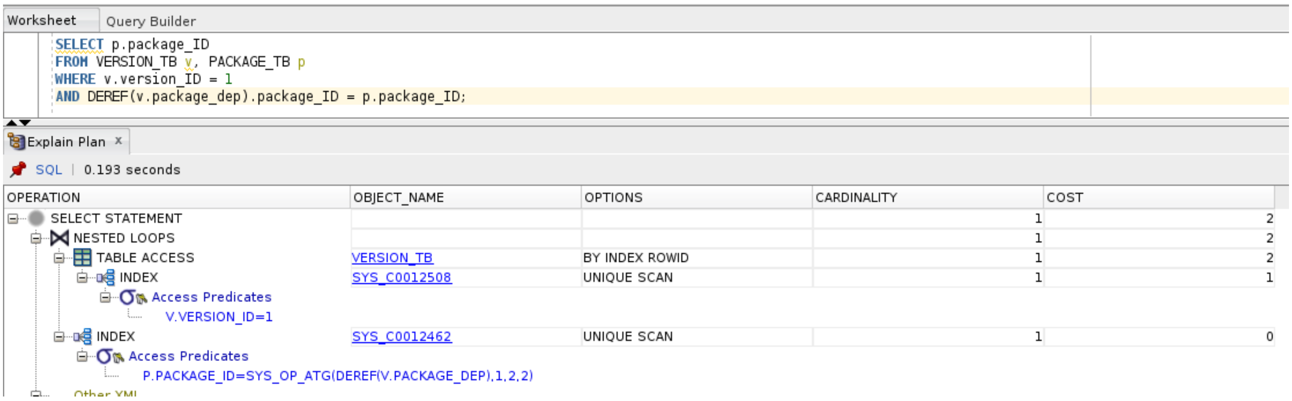
```
END;
```

QUERY OPTIMIZATION

In order to optimize the operations in our database, we rewrite each query that we use in our procedures, and we see the output of Explanation Plan which is available in SQL Developer.

Operation 1

For this operation we used two queries. The first query is needed to get the package id by using the id of the version we want to install.

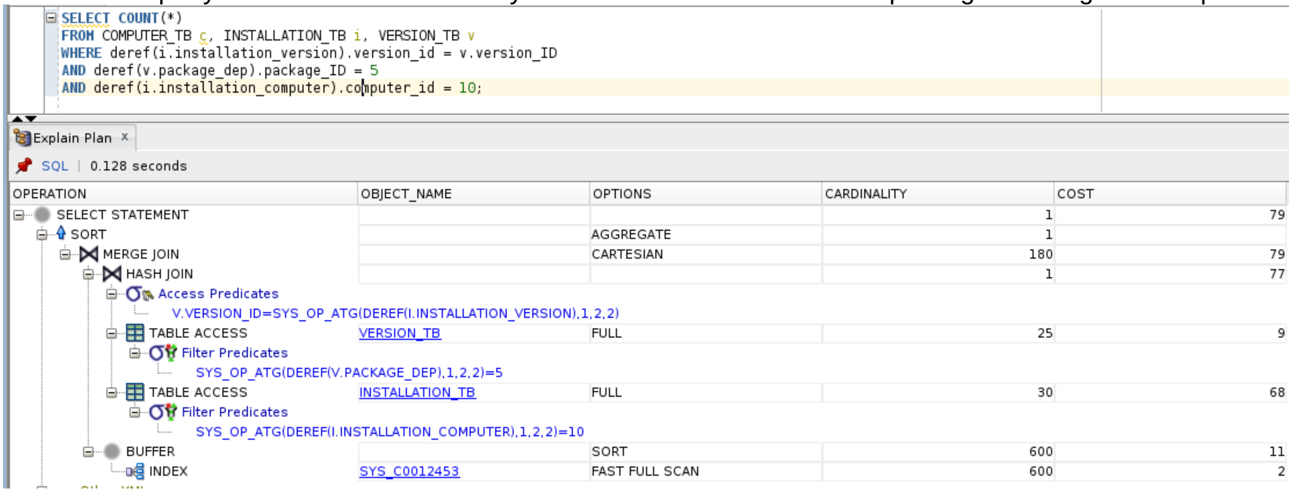


```
SELECT p.package_ID
FROM VERSION_TB v, PACKAGE_TB p
WHERE v.version_ID = 1
AND DEREF(v.package_dep).package_ID = p.package_ID;
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	2
NESTED LOOPS			1	2
TABLE ACCESS	VERSION_TB	BY INDEX ROWID	1	2
INDEX	SYS_C0012508	UNIQUE SCAN	1	1
Access Predicates				
V.VERSION_ID=1				
INDEX	SYS_C0012462	UNIQUE SCAN	1	0
Access Predicates				
P.PACKAGE_ID=SYS_OP_ATG(DEREF(V.PACKAGE_DEP),1,2,2)				
Other XML				

The query is very simple, and the cost is very low, also because is performed on already indexed attributes.

The second query check if there are already installed versions of the same package on the given computer.



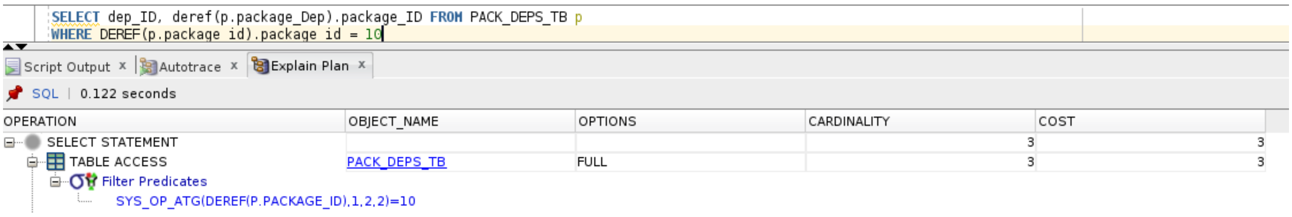
```
SELECT COUNT(*)
FROM COMPUTER_TB c, INSTALLATION_TB i, VERSION_TB v
WHERE deref(i.installation_version).version_id = v.version_ID
AND deref(v.package_dep).package_ID = 5
AND deref(i.installation_computer).computer_id = 10;
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	79
SORT		AGGREGATE	1	
MERGE JOIN		CARTESIAN	180	79
HASH JOIN			1	77
Access Predicates				
V.VERSION_ID=SYS_OP_ATG(DEREF(I.INSTALLATION_VERSION),1,2,2)				
TABLE ACCESS	VERSION_TB	FULL	25	9
Filter Predicates				
SYS_OP_ATG(DEREF(V.PACKAGE_DEP),1,2,2)=5				
TABLE ACCESS	INSTALLATION_TB	FULL	30	68
Filter Predicates				
SYS_OP_ATG(DEREF(I.INSTALLATION_COMPUTER),1,2,2)=10				
BUFFER		SORT	600	11
INDEX	SYS_C0012453	FAST FULL SCAN	600	2

Although a join is performed on three tables, the query is quite efficient since refs are used and the computer id attribute is already indexed being a primary key. We do not need to optimize this query.

Operation 2

The first query in the second procedure is needed to get the required dependencies for the package that we want to install.

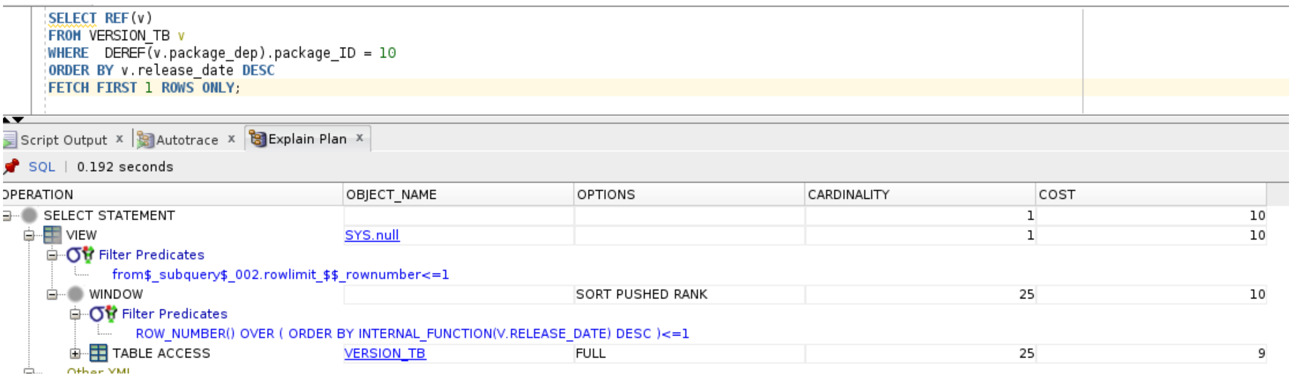


```
SELECT dep_ID, deref(p.package_Dep).package_ID FROM PACK_DEPS_TB p
WHERE DEREF(p.package_id).package_id = 10
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			3	3
TABLE ACCESS	PACK_DEPS_TB	FULL	3	3
Filter Predicates				
SYS_OP_ATG(DEREF(P.PACKAGE_ID),1,2,2)=10				

We don’t need to optimize the query, is very simple.

The second query is also very simple.
The third query is the following one.

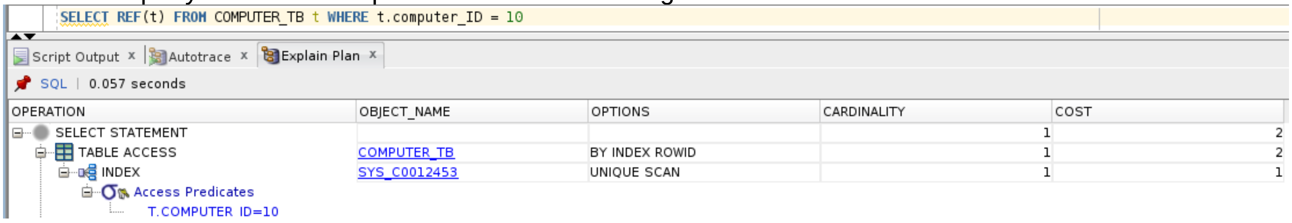


```
SELECT REF(v)
FROM VERSION_TB v
WHERE  DEREF(v.package_dep).package_ID = 10
ORDER BY v.release_date DESC
FETCH FIRST 1 ROWS ONLY;
```

PERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	10
VIEW	SYS.null		1	10
Filter Predicates				
from$_subquery$_002.rowlimit_$$_rownumber<=1				
WINDOW		SORT PUSHED RANK	25	10
Filter Predicates				
ROW_NUMBER() OVER (ORDER BY INTERNAL_FUNCTION(V.RELEASE_DATE) DESC)<=1				
TABLE ACCESS	VERSION_TB	FULL	25	9
Other XML				

We can see that this query can optimized because the order by is performed on an attribute that is not indexed.
After we create an index on release date attribute we see again the execution plan and the cost remains the same, but the execution time of the same query decrease from 0.7 second to 0.3 second, but can not be related.

The fourth query in the second procedure is the following.



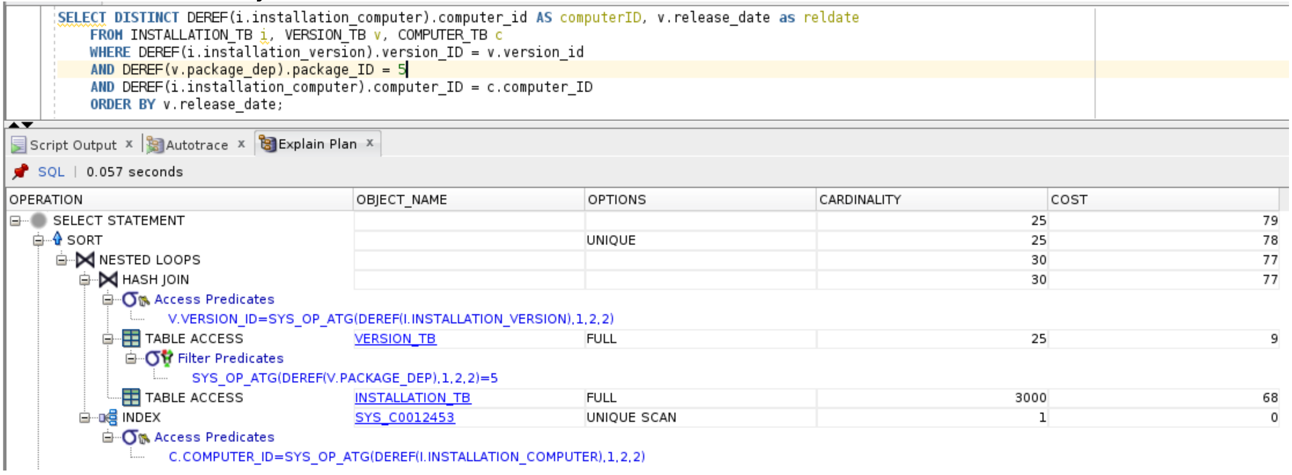
```
SELECT REF(t) FROM COMPUTER_TB t WHERE t.computer_ID = 10
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	2
TABLE ACCESS	COMPUTER_TB	BY INDEX ROWID	1	2
INDEX	SYS_C0012453	UNIQUE SCAN	1	1
Access Predicates				
T.COMPUTER_ID=10				

Is very simple query, so we do not need to optimize.

Operation 3

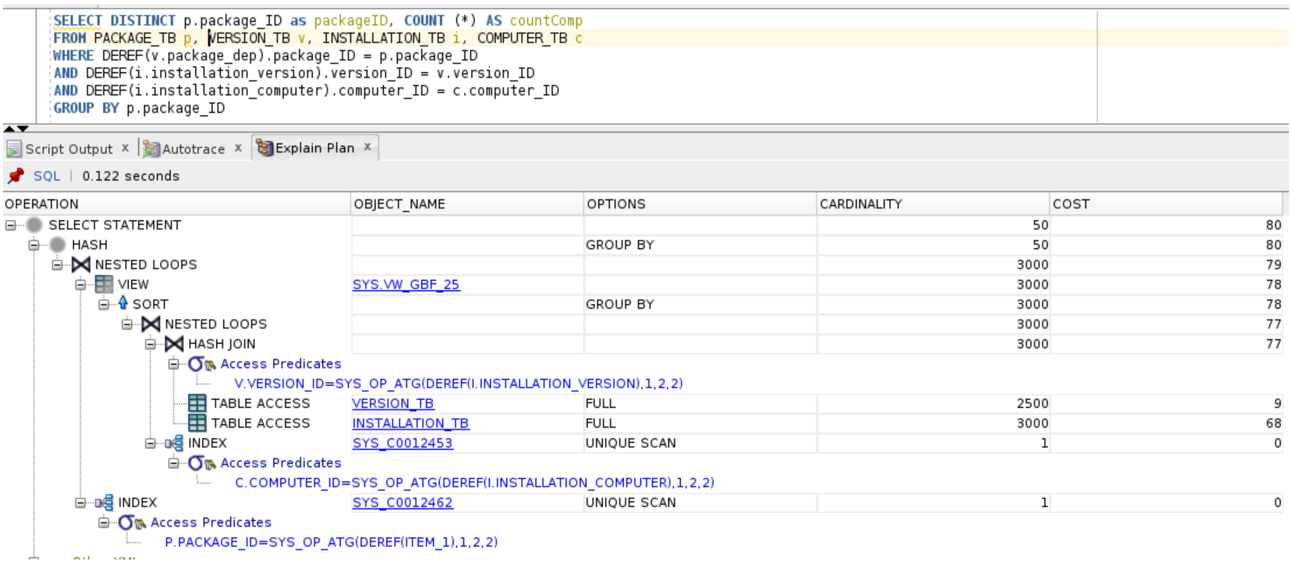
The operation 3 requires the use of two query, one is very simple (just used to count how many computers we have for the given package) and the other one take the list of the computer on which the given package is installed, ordered by the release date of the installed version



```
SELECT DISTINCT DEREF(i.installation_computer).computer_id AS computerID, v.release_date as reldate
    FROM INSTALLATION_TB i, VERSION_TB v, COMPUTER_TB c
    WHERE DEREF(i.installation_version).version_ID = v.version_id
    AND DEREF(v.package_dep).package_ID = 5
    AND DEREF(i.installation_computer).computer_ID = c.computer_ID
    ORDER BY v.release_date;
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			25	79
SORT		UNIQUE	25	78
NESTED LOOPS			30	77
HASH JOIN			30	77
Access Predicates				
V.VERSION_ID=SYS_OP_ATG(DEREF(I.INSTALLATION_VERSION),1,2,2)				
TABLE ACCESS	VERSION_TB	FULL	25	9
Filter Predicates				
SYS_OP_ATG(DEREF(V.PACKAGE_DEP),1,2,2)=5				
TABLE ACCESS	INSTALLATION_TB	FULL	3000	68
INDEX	SYS_C0012453	UNIQUE SCAN	1	0
Access Predicates				
C.COMPUTER_ID=SYS_OP_ATG(DEREF(I.INSTALLATION_COMPUTER),1,2,2)				

We can decrease the cost of such a query because and ordered is performed on a non-index attribute (rel_date).
For this reason we created and index on release date, but after we create it we do not see any improvement.
In this operation we use only one query which is the following one.



```
SELECT DISTINCT p.package_ID as packageID, COUNT (*) AS countComp
FROM PACKAGE_TB p, VERSION_TB v, INSTALLATION_TB i, COMPUTER_TB c
WHERE DEREF(v.package_dep).package_ID = p.package_ID
AND DEREF(i.installation_version).version_ID = v.version_ID
AND DEREF(i.installation_computer).computer_ID = c.computer_ID
GROUP BY p.package_ID
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			50	80
HASH		GROUP BY	50	80
NESTED LOOPS			3000	79
VIEW	SYS.VW_GBF_25		3000	78
SORT		GROUP BY	3000	78
NESTED LOOPS			3000	77
HASH JOIN			3000	77
Access Predicates				
V.VERSION_ID=SYS_OP_ATG(DEREF(I.INSTALLATION_VERSION),1,2,2)				
TABLE ACCESS	VERSION_TB	FULL	2500	9
TABLE ACCESS	INSTALLATION_TB	FULL	3000	68
INDEX	SYS_C0012453	UNIQUE SCAN	1	0
Access Predicates				
C.COMPUTER_ID=SYS_OP_ATG(DEREF(I.INSTALLATION_COMPUTER),1,2,2)				
INDEX	SYS_C0012462	UNIQUE SCAN	1	0
Access Predicates				
P.PACKAGE_ID=SYS_OP_ATG(DEREF(ITEM_1),1,2,2)				

The query is quite efficient because already uses indexed attributes and references, which are efficient.
Also, this procedure is invoked only 16 times per day.
We do not optimize this query.

Operation 5

This operation requires only delete on installation table and versions table, and is made only 15 times per day, so we do not optimize this query.