ESTÁNDAR BASE DE DATOS ORACLE V5.1

Contenido

1. Control de cambios

Nombre	Observación	Fecha
OSCAR CARDOZO	Redacción y publicación	26 de julio de 2014
DANIEL CALDERON	Revisiones generales, actualización módulo de optimización, nueva esquematización del documento, adhesión de ejemplos prácticos.	23 de octubre de 2014
OSCAR CARDOZO	Revisiones generales, actualización módulo de optimización y adecuación temas de Paso a producción	20 de febrero de 2015
OSCAR MOYA	Revisiones generales, actualización, adhesión de nuevas prácticas.	02 de Junio de 2016
OSCAR MOYA	Adhesión de nuevas prácticas con apoyo del documento “Estandares de desarrollo y entregables para Oracle” versión 1. De la Gerencia de Bases de Datos y Middleware Claro. Nueva esquematización.	16 de agosto de 2016
OSCAR CARDOZO	Revisiones generales, actualización módulo de optimización y adecuación temas de Paso a producción	13 de diciembre de 2016
OSCAR CARDOZO	Aclaraciones temas de permisos en script de seguridad en PAP	20 de diciembre de 2016
CARLOS BARRIOS	Manejo de spool en los scripts de ejecuciones.	24 de abril de 2019

2. ¿A quién va dirigido?

Este estándar está dirigido a quienes su SGBD esté soportado sobre Oracle Database 10g y 11gR2 y que tengan participación dentro del proceso dado entre el diseño de la base de datos, su paso a producción y mantenimiento.

Para el correcto entendimiento de este documento se requieren conocimientos en el lenguaje SQL, Modelo Entidad Relación y objetos de base de datos. Sin embargo, como material de apoyo, se tiene a disposición la documentación de fabricante para Oracle 11gR2 http://docs.oracle.com/cd/E11882_01/index.htm y la guía de referencia SQL http://docs.oracle.com/cd/E11882_01/server.112/e41084/toc.htm

3. Introducción

Este documento hace referencia a instrucciones básicas de nomenclatura, diseño, optimización y otros aspectos de los diferentes objetos en la base de datos para flexibilizar y mejorar su implementación, desarrollo y administración.

No todos los objetos que componen una base de datos están contemplados en este documento, no obstante, se generan revisiones y actualizaciones progresivamente.

También hace referencia a los elementos más comunes y buenas prácticas en la optimización de consultas SQL sobre el motor de base de datos Oracle 11gR2 y se construye con el apoyo de la guía de optimización del fabricante y recomendaciones de expertos en la materia.

El alcance de este instructivo tiene en cuenta el contexto fundamental del CBO (Optimizador Basado en Costo –sigla en inglés) y las recomendaciones aplicables a los scripts con el propósito de buscar el mejor rendimiento desde el primer despliegue en producción.

Para varios ejemplos se toma como referencia el esquema HR compartido por Oracle para desarrollo y aprendizaje.

4. Nombramiento de objetos

4.1 Prefijos

Todos los objetos de la base de datos deben tener un prefijo identificador, este no debe de pasar de los 5 caracteres y debe finalizar con el caracter underline: "_"y que dependerán de la aplicación o módulo que corresponda. Por ejemplo si los objetos son para el Portal de Contenidos el prefijo podría ser "PCT_"; la funcionalidad de este prefijo es poder localizar los objetos utilizando filtros por nombre y usar el diccionario de datos para identificación, valoración e incluso exportación de una manera más ágil.

ESTRUCTURA	EJEMPLO
`CREATE <objeto> <esquema>.<prefijo><objeto><sufijo> ...`	`CREATE INDEX GESTIONNEW.AGEN_AGEN_IX ...`

Casos especiales de prefijos

La identificación de Tablespaces deben comenzar con la identificación del objeto, por ejemplo TBL_PRY_DT, el cual está compuesto por la identificación de una tablespace (TBL), el prefijo de proyecto (PRY) y el sufijo que indica el destino (DT –Datos/índices), todo separado por el carácter "_".

4.2 Sufijos

Los objetos de la base de datos deben tener un sufijo identificador, con el fin de diferenciar el tipo de objeto creado, la excepción a la regla son las tablas (a menos que sean de auditoría) y las vistas; este debe comenzar con el caracter underline: "_", se recomienda:

Objetos	Sufijos		Ejemplo
Tabla*	TB		PRY_PRUEBA
Tabla auditoria	$AUD		PRY_PRUEBA_$AUD
Tabla temporal	TMP		PRY_PRUEBA_TMP
Tabla histórica	HST	HIST	PRY_PRUEBA_HST
Llave primaria	PK		PRY_PRUEBA_PK
Índices	IX	IDX	PRY_PRUEBA_IX
Registro único	UK		PRY_PRUEBA_UK
Restricción	CK		PRY_PRUEBA_CK
Llave foránea	FK	FRK	PRY_PRUEBA_FK
Función	FN		PRY_PRUEBA_FN
Procedimiento	SP		PRY_PRUEBA_SP
Paquetes	PKG		PRY_PRUEBA_PKG
Trigger	TR	TRG	PRY_PRUEBA_TRG
Secuencias	SQ	SEQ	PRY_PRUEBA_SQ
Vistas*	VW		PRY_PRUEBA_VW
Tablespace de datos	DT		TBL_PRY_DT
Tablespace de índices	IX		TBL_PRY_IX
Particiones	PRT		PRY_PRUEBA_PRT
Job	JB		PRY_PRUEBA_JB

**Objetos cuyo sufijo es opcional.*

Casos especiales de sufijos

En el caso de objetos como índices, llaves foráneas, particiones, donde puede preverse que se duplique el nombre, se puede finalizar con un indicador numérico, por ejemplo PRY_PRUEBA_IX1 y PRY_PRUEBA_IX2, donde son índices de la misma tabla pero que tienen en cuenta diferentes campos, funciones, etc.

4.3 Nomenclatura

Hay algunos aspectos que son comunes a la hora de establecer la nomenclatura de los objetos de la base de datos:

- El nombre de un objeto no debe superar los 30 caracteres, teniendo en cuenta los prefijos y sufijos descritos anteriormente.
- El nombre del objeto debe estar separado por el carácter "_". Ver los ejemplos anteriormente descritos.
- El nombre ubicado entre el prefijo y sufijo debe ser diciente o claro con respecto al uso, la mejor manera de explicar este término es a través de las llaves foráneas donde el nombre debería por lo menos indicar los campos a relacionar o tener un nombre que sea explícito para identificar la relación.
- Los únicos caracteres especiales permitidos con el "_" y el "$", con el primero se determina la separación y con el segundo se identifica que es una tabla de auditoría. Lo mismo aplica para los nombres de los campos de una tabla.

5. Consideraciones por objetos

5.1 Tablas

- **Propiedades generales:**
  - Tanto la tabla como los campos deben tener sus respectivos comentarios.
  - La tabla debe ser enlazada al tablespace definido para los datos y los índices al definido para este aspecto.
  - Las tablas de alta transaccionalidad deben ir asociadas a TABLASPACE de gran tamaño y las de baja transaccionalidad a TABLESPACE de tamaño bajo y medio, igualmente sus índices.
  - Todos los campos deben ser NOT NULL y deben ir con un valor por DEFAULT; las únicas excepciones son la llave primaria que no debe tener un valor por defecto y los campos de fecha de actualización que pueden ir nulos.
  - Todas las tablas deben tener mínimo un campo fecha tipo DATE, y si es una tabla transaccional debe contener el campo fecha de inserción del registro y mínimo un campo más que almacene la fecha de modificación o actualización.

- Las tablas deben tener índices adicionales al de la PRIMARY KEY pare mejorar rendimiento, considerando las columnas de uso más frecuente en las clausulas WHERE.
- Los campos como estado o que tengan valores fijos, deben tener la respectiva restricción por tabla.

- **Campo Auto-numérico**:
  - Debe estar referenciado a una secuencia y a un trigger a menos que se maneje desde la capa de aplicación y debe ser aclarado como un comentario en el PAP.
  - El campo debe ser tipo NUMBER de 15 dígitos.
  - Su nombre no debe ser genérico como “ID”, sino que debe nombrarse con mayor especificación, por ejemplo: “USUARIO_ID” .
  - Si el atributo pertenece a una llave foránea, los campos de origen y destino deben tener el mismo nombre, tipo y longitud de datos.
- **Longitud de atributos:**
  - Adecuar la longitud del atributo a la información a almacenar procurando ser lo más conservador posible. Por ejemplo, el uso de un VARCHAR2(2000) no se justifica para un campo que contenga un nombre propio o la dirección de un domicilio.
  - No se recomienda el uso de campos LOB, si se emplean deber estar autorizados por el área de BD, realizando el respectivo sizing.
- **Llave foránea:**
  - Si la tabla contiene llaves foráneas, el campo que tenga este atributo, debe tener las mismas características del mismo campo de la tabla padre.
  - Se debe crear Índice para el campo asociado de la tabla hija.
- **Coherencia de la data con el DATA TYPE:**
  - Para los atributos con información numérica, se debe usar un TIPO NUMBER especificando la cantidad de decimales si los tiene.
  - Para los campos que identifiquen un tipo de estado, por ejemplo ACTIVO e INACTIVO, se recomienda el uso de campos numéricos con solo un digito con restricción de dominio (que solo permita elementos válidos como 1 o 0).
  - Teniendo en cuenta el punto anterior, de no ser posible implementar valores numéricos se recomienda un tipo CHAR con restricción para los caracteres “A” e “I” o según corresponda.
  - Si un atributo contiene información de FECHA/HORA se recomienda el uso de DATE, no obstante, si se requiere los motores de datos tienen tipos de fecha TIMESTAMP que se rigen por la norma ISO 8601.
  - Hay que tener cuidado con campos de tipo especial como los LOB, que deben ser consultados con el DBA ya que su uso puede implicar un aumento considerable en el tamaño del objeto y el posible uso de un tablespace independiente para optimizar su manejo.
  - Los campos de CLOB, BLOB etc. Debe aplicárseles por default la función EMPTY_CLOB() o EMPTY_BLOB() respectivamente, con el fin de proporcionar un valor de longitud cero a dichos campos para cuando éstos estén vacíos.

- **Tablas de auditoría:**
  - Aplican las instrucciones anteriores y, adicionalmente,
  - Validar que el trigger solo contemple las acciones de actualización y eliminación, ya que la inserción se encuentra en la tabla padre.
  - Solicitar al DBA o a quien corresponda el agendamiento de procesos de limpieza de datos en un intervalo de máximo cada 6 meses (*Obligatorio)*.
- **Recomendaciones tablas históricas.**
  - Se recomienda la adición de un auto numérico para estas tablas o la adecuación de una llave primaria contra un campo de fecha automático.
  - Para el momento de generar las particiones se recomienda el uso de un campo de fecha preferiblemente automático.
  - Si se particiona una tabla que tenga relaciones, todas las tablas hijas deberían tener partición (según sea el análisis).

5.2 Variables y constantes

- Para declarar una variable se debe usar como nomenclatura V_<nombre_variable>.
- Las variables que hagan alusión a un campo de una tabla existente, se deben declarar con el %TYPE para que tomen las mismas características del campo, anteponiendo siempre el esquema en la tabla. Por Ejemplo:

```
V_IDAGENDA GESTIONNEW.AGENDA.IDAGENDA%TYPE;
```

- Evite la reutilización de variables, cada variable que declare debe tener un propósito único.
- El nombre para esa variable debe describir, tan claramente como sea posible, su único propósito.
- Para declarar una constante ya sea dentro de una función o procedimiento almacenado u otro tipo de objeto de este estilo se debe usar como nomenclatura C_<nombre_constante>.
- Quite todos los "números mágicos" y otros literales (dentro de lo razonable) de su código.
- Establezca el valor de ese literal (constante) en un solo lugar en el código.
- Si se encuentra que se ha escrito un programa en el que un valor de la variable no cambia, en primer lugar debería determinar si ese comportamiento es correcto. Si es así, se debe entonces convertir esa variable a una constante, por lo que debe también cambiar su nombre. Esto le ayudará a recordar en la lectura del código que su identificador se refiere a una constante y es inalterable.
- Se deben revisar los programas periódicamente y eliminar cualquier parte del código que ya no se utiliza. Este es un proceso relativamente sencillo para las variables y constantes con nombre. Basta con ejecutar una búsqueda con el nombre de esa variable en el ámbito de la aplicación, si se encuentra que el único lugar que aparece es en la declaración, elimine la declaración.

6. Funciones y procedimientos almacenados.

- Una función sólo puede devolver una variable, de modo que si se necesita devolver dos o más valores (tal vez olvidándose de los registros, tipos de objetos y colecciones), el uso de múltiples parámetros es la solución más lógica, aunque cualquiera que haga mantenimiento del código tiene que averiguar cuáles de los parámetros son de “entrada” y cuáles de “salida”, para comprobar esto, se tendrá que encontrar la declaración del procedimiento, lo cual puede ser difícil de mantener en sistemas complejos, por lo cual la documentación juega un papel importante para ayudar a resolver y simplificar los procesos.
- No agregue paréntesis redundantes. Observe que PL / SQL proporciona palabras clave para poner fin a las expresiones condicionales. Por ejemplo: IF x = 1 AND y = 2 THEN.
- Se pueden utilizar palabras clave en la misma línea siempre y cuando la condición sea corta y simple, o puede iniciar una nueva línea directamente por debajo del condicional (IF).
- Del mismo modo, cuando se tiene más de una condición, si sólo hay dos o tres y son muy sencillas las puede colocar a todas en una sola línea, de lo contrario es recomendable escribir una línea por cada condicional, alineadas verticalmente: IF TO_CHAR(SYSDATE,'D') > 1 AND MYPACKAGE.MYPROCEDURE(1,2,3,4) NOT BETWEEN 1 AND 99 THEN.
- Cuando hay una combinación de condiciones y, a menos que sean muy simples se recomienda utilizar espacios para alinear verticalmente entre paréntesis: IF a = 1 AND (b = 2 OR c = 3 OR d = 4) THEN.
- No utilice EXIT o RETURN fuera de in ciclo FOR o WHILE
- Un bucle FOR sólo debe utilizarse cuando se desea ejecutar un código un número determinado de veces.El bucle debe darse por concluido sólo cuando la condición evaluada es FALSE.
- Hay que tener cuidado al utilizar los nombres de las excepciones, en particular, un programador puede definir una excepción y con las excepciones definidas por el motor de la base de datos se puede buscar el código en un manual, pero con los códigos creados internamente la documentación debe ser lo suficientemente clara para determinar por qué y dónde es lanzada la excepción.
- La sentencia que tiene internamente debe estar optimizada lo mejor posible.
- Si se realizan inserciones, actualizaciones o eliminación, deben estar dentro de un cursor o ciclo para optimizar el proceso.
- Utilice el carácter de tabulación (no espacios) como la unidad de sangría para los bloques de código PL / SQL, no use espacios a la izquierda o una mezcla de tabuladores y espacios o cualquier otra cosa.
- Los objetos deben tener el siguiente encabezado:

```
/*
Nombre: Nombre del programa u objeto.
Objetivo: Objetivo general del programa
Realizado Por: Nombre de la empresa separada por un guion del nombre del desarrollador.
Fecha: Fecha
Modificación1: Nombre persona que modifica, fecha y descripción.
Modificacion2: Nombre persona que modifica, fecha y descripción.
Modificacion(n): Nombre persona que modifica, fecha y descripción.
*/
```

- Los objetos deben contener en el cuerpo:

```
/* Descripción de la funcionalidad de cada ciclo*/

        Begin
          /* Descripción o funcionalidad*/
         Loop
            /* Descripción o funcionalidad*/
           While
              ……..
           End While
         End Loop
        End If
```

7. Paquetes

- Se recomienda el uso de paquetes para encapsular las funciones y procedimientos, esto siempre y cuando sea un esquema compartido por varias aplicaciones y se necesite identificar y separar los procesos del mismo; en caso que sea un esquema propio se recomienda el uso individual de los objetos.
- El orden de encapsulamiento de los objetos debe ser según uso de los mismos, primero funciones y luego procedimiento almacenados.
- Los objetos deben tener el siguiente encabezado:

```
/*
Nombre: Nombre del programa u objeto.
Objetivo: Objetivo general del programa
Realizado Por: Nombre de la empresa separada por un guion del nombre del desarrollador.
Fecha: Fecha
Modificación1: Nombre persona que modifica, fecha y descripción.
Modificacion2: Nombre persona que modifica, fecha y descripción.
Modificacion(n): Nombre persona que modifica, fecha y descripción.
*/
```

- Los objetos deben contener en el cuerpo:

```
/* Descripción de la funcionalidad de cada ciclo*/

        Begin
          /* Descripción o funcionalidad*/
         Loop
            /* Descripción o funcionalidad*/
```

```
While
    ……..
End While
End Loop
End If
```

8. Secuencias

- Deben siempre inicializar en 1.
- Deben tener el máximo número final posible (según motor de base de datos).
- Deben estar en cache de memoria, por defecto un valor de 20, no obstante si el análisis de utilización de esta secuencia es alto este valor debe aumentar según recomendación del DBA.
- Evitar el uso de CURRVAL en transacciones DML, en su lugar emplear NEXTVAL. En ambientes de concurrencia el uso de CURRVAL puede generar errores de duplicidad.

9. Vistas

- Los objetos deben tener el siguiente encabezado:

```
/*
Nombre: Nombre del programa u objeto.
Objetivo: Objetivo general del programa
Realizado Por: Nombre de la empresa separada por un guion del nombre del desarrollador.
Fecha: Fecha
Modificación1: Nombre persona que modifica, fecha y descripción.
Modificacion2: Nombre persona que modifica, fecha y descripción.
Modificacion(n): Nombre persona que modifica, fecha y descripción.
*/
```

- La sentencia que tiene internamente debe estar optimizada lo mejor posible.

10. Otros objetos

- Objetos como Job, tipos nuevos, entre otros deben ser consultados con los DBA, para validar si su creación está permitida o si es óptima.
- Este tipo de objeto ya tiene su propio estándar así que no se aplica nueva reglas, que no sean definidas por el DBA.

11. Paso a producción

- Es de carácter obligatorio que en los scripts enviados, las sentencias para crear, modificar o eliminar estructuras, Jobs, procedimientos almacenados, paquetes, funciones, índices, etc. se incluya el esquema de acuerdo a la sintaxis especificada anteriormente.
- En todo script de manipulación de datos las sentencias de selección, actualización, eliminación, adición de datos, etc. Se debe incluir el esquema de acuerdo a la sintaxis especificada anteriormente.
- Es de carácter obligatorio que en los scripts que incluyen creación de paquetes, funciones, trigger y paquetes, etc. cada objeto debe terminar con el símbolo /.
- Para todo script de tablas, vistas y demás se deben incluir también los sinónimos, secuencias, restricciones etc. (no en script separados, Se deben agrupar la mayoría de operaciones.)
- Debe existir siempre en el documento de paso a producción como precondición, la generación de backups de los objetos específicos que se pretenden modificar.
- Utilice el carácter de tabulación (no espacios) como la unidad de sangría para los bloques de código PL / SQL, no use espacios a la izquierda o una mezcla de tabuladores y espacios o cualquier otra cosa.
- Para identificar plenamente el objeto debe venir descrito de la siguiente manera: ESQUEMA.NOMBREOBJETO.
- Los siguientes archivos deben tener como primera línea:
    - ALTER SESSION SET CURRENT_SCHEMA= “NOMBREDEL ESQUEMA”;
- Ninguna aplicación debe eliminar registros, solo debe cambiar de estado; si la funcionalidad exigida de la aplicación solicita esta funcionalidad, es obligatorio la creación de la tabla de auditoría y el log de registro de eliminación y debe ser aclarado antes de presentar el PAP.
- Si se toca algún objeto preexistente, es obligación aplicar el estándar y realizar la optimización correspondiente.
- Lo objetos deben ser creados en el esquema propietario, pero en el archivo de seguridad se deben dar los permisos al usuario de aplicación el cual debió ser solicitado con anterioridad.

Organización del contenido de los Objetos:

- El contenido de los objetos debe ir organizado de la siguiente manera:

1. Objetos.sql

- ALTER SESSION al esquema propietario
- Secuencias
- Tablas
    - Comentarios para cada tabla.
    - Índices para cada tabla.
- Triggers finalizados con el operador /
- Sinónimos.

12. Deshacer.sql

- ALTER SESSION al esquema propietario.
- Sinónimos
- Secuencias
- Tablas
- Procedimientos , funciones (si aplica)
- Paquetes

Para los puntos 4 y 5 si ya existía, se debe dejar el código anterior.

Orden de Ejecución:

- Para los pasos a producción y cambios emergentes siempre se debe indicar el orden de ejecución de los scripts, los archivos serán organizados de la siguiente manera:
  - 1. Objetos.sql: script de creación y/o alteración de tablas, índices, otros objetos inherentes a las tablas, secuencias y sinónimos.
    - En los trigger después de la cláusula ON (donde va la tabla), también debe ir el esquema
  - 2. Procedimientos.sql: script de creación y/o alteración de procedimientos, funciones y sus sinónimos.
  - 3. Vistas.sql: script de creación y/o alteración de vitas y sus sinónimos.
  - 5. Job.sql; script de creación y/o alteración de Job y sus sinónimos
  - 6. Paquete.sql: script de creación y/o alteración de paquetes y sus sinónimos.
  - 7. Colas: script de creación y/o alteración.
  - 11. Seguridad.sql: permisos requeridos por objetos (se aplican sobre roles, no usuarios).
    - Además de asignar el permiso al usuario de aplicación se debe tener en cuenta:
      - Para secuencias solo debe ir el permisos de select
      - Para tablas deben ir permisos de select, insert y update
        - Si es en la base GESTION se deben asignar permisos de select a los roles CONSULTA_GESTION,CONSULTA_ESQUEMAS
        - Si es en la base GESTION se deben asignar permisos de select, insert y update al rol MODDATA_GESTION
      - Para las vistas solo debe ir el permisos de select
        - Si es en la base GESTION se deben asignar permisos de select a los roles CONSULTA_GESTION,CONSULTA_ESQUEMAS y MODDATA_GESTION
      - Para las funciones, procedimientos y paquetes solo debe ir el permisos de execute
        - Si es en la base GESTION se deben asignar permisos de execute al rol MODDATA_GESTION

        - Si hay otro tipo de objetos deben ser escaldos a Oracle los tipos de permisos
    - 12. Deshacer.sql: comandos de rollback, para los script del 1 al 7.
- Como se puede ver no va el archivo 4, que corresponde a los datos a insertar, actualizar o borrar, dichos datos son responsabilidad del área que solicita el paso a producción.
- Se debe tener en cuenta en el archivo 11 la asignación de permisos, se debe consultar con el DBA si hay roles y sus respectivos privilegios, además también se verificaría la necesidad de crear sinónimos públicos.
    - Todo objeto de la base de producción en su esquema debe tener sinónimo público, la única excepción para para esquemas que no comparten información con ningún otro dentro de la misma base
- Se debe adicionar la evidencia de ejecución correcta de los script tanto de creación como de rollback, en los ambientes de pruebas, como evidencia del funcionamiento óptimo de los script.
- Se debe enviar el dimensionamiento de la base de datos, con el fin que el área encargada pueda validar si existe el espacio para alojar los objetos y futuros datos
- El documento donde se va a indicar las actividades a realizar será el PAP, este documento se va a dejar en una carpeta diferente a todo el resto de documentación.
- En el PAP se deben tener en cuenta:
    - En el ítem SERVIDOR: indicar base de datos y servidor.
    - En el ítem Precondiciones: por ejemplo el backup de tablas específicas.
    - En el ítem actividades: actividad, y quién ejecuta y cuándo.
- Por temas de documentación se debe enviar el modelo entidad relación de la base de datos.
- Por temas de documentación se debe enviar el Diccionario de datos.

12. Optimización de sentencias y otras consideraciones

12.1 Optimización basada en Costo (CBO)

El Optimizador basado en costo (CBO – Cost Based Optimizer) es un componente Oracle que regula la ejecución de todas las consultas SQL. Se ha convertido en uno de los más avanzados componentes a nivel mundial y tiene el reto de evaluar cualquier sentencia SQL y generar "el mejor" plan de ejecución.

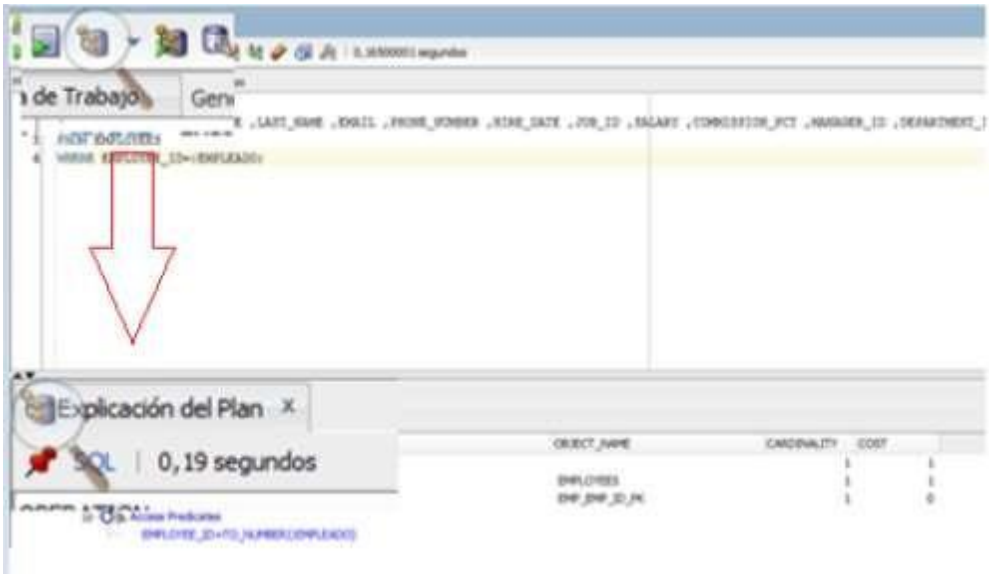


Figura 1 Optimización Basada en Costo.

La cifra reportada en el "*Cost*" de Oracle SQL Developer debe tender a 1 para todas las operaciones generadas y/o al mínimo de operaciones posibles.

Entre versiones distintas de Oracle Database, el plan de ejecución puede cambiar, teniendo en cuenta que tiene nueva información disponible (mejoras y características propias de la versión), rutas de acceso, indicios (Hints), estadísticas, etc., y selecciona aquel plan que genere el menor costo.

12.2 Optimización a partir del diseño de la DB.

El impacto a la Base de Datos puede reducirse desde el diseño en la definición de nombres, data types, etc., así:

- Nombramiento de atributos. Evitar nombres de tabla demasiado extensos y con caracteres especiales (a excepción de prefijos, sufijos y tablas de auditoría) recordando que el tamaño máximo es de 30 caracteres.
- Uso de NULL. Evitar el uso de valores NULL y procurar un valor Default de mínimo tamaño. Teniendo en cuenta que los índices omiten los valores nulos y aquellas consultas que usan cláusulas IS [NOT] NULL, requerirían la creación de otro tipo de índices especiales cuya funcionalidad puede verse limitada tras cambios posteriores.
- Uso correcto de tipos de datos. Los tipos de datos fueron concebidos teniendo en cuenta no sólo el tipo de información que se ingresa según el contexto (numérica, caracter) sino por la manera en cómo se almacena y es tratada para las operaciones. Dentro del diseño de la base de datos es importante otorgar a cada campo el tipo de dato que realmente le corresponda. Por ejemplo un número de tres dígitos como "456" consume 2 Bytes si se almacena como tipo NUMBER y 3 Bytes como VARCHAR2 comprometiendo espacio en disco y, por supuesto, la manera óptima de realizar cálculos, conversiones, etc. De igual

manera sucede con el almacenamiento de fecha/hora, si se almacena como como DATE y no como tipo carácter, podrá costar casi la mitad en resolver las consultas asociadas y la estimación del optimizador es mucho más acertada.

- Verificar que los campos atributos relacionados entre tablas (MER) tengan la misma estructura de datos y validación NOT NULL para evitar conversiones en consultas de unión.

12.3 Uso de funciones Analíticas

El uso correcto de funciones analíticas permiten disminuir la cantidad de entradas a una tabla representadas en Self Joins (uniones de la misma tabla) o productos cartesianos (tablas separadas por comas).

Ejemplo:

Se desea obtener un listado comparativo de empleados cuyo salario sea mayor al promedio del departamento o al promedio del cargo.

Sentencia 1:

```
SELECT
  FIRST_NAME NOMBRE, SALARY SALARIO,DEPARTMENT_ID DEPARTAMENTO,JOB_ID CARGO,
  (SELECT ROUND(AVG(SALARY),0)FROM EMPLOYEES WHERE DEPARTMENT_ID=EMPL.DEPARTMENT_ID)PROMEDIO_DEPTO,
  (SELECT ROUND(AVG(SALARY),0)FROM EMPLOYEES WHERE JOB_ID=EMPL.JOB_ID)PROMEDIO_CARGO
FROM EMPLOYEES EMPL
WHERE SALARY > (SELECT AVG(SALARY)PROMEDIO_DEPTO FROM EMPLOYEES WHERE DEPARTMENT_ID = EMPL.DEPARTMENT_ID)
      OR SALARY > (SELECT AVG(SALARY)PROMEDIO_CARGO FROM EMPLOYEES WHERE JOB_ID = EMPL.JOB_ID)
ORDER BY 2 DESC;
```

Resultado:

	NOMBRE	SALARIO	DEPARTAMENTO	CARGO	PROMEDIO_DEPTO	PROMEDIO_CARGO
1	Steven	24000	90	AD_PRES	19333	24000
2	Michael	13000	20	MK_MAN	9500	13000
3	Shelley	12000	110	AC_MGR	10150	12000
4	Ellen	11000	80	SA_REP	10033	8867
5	Eleni	10500	80	SA_MAN	10033	10500
6	Alexander	9000	60	IT_PROG	6400	6400
7	Kevin	5800	50	ST_MAN	3500	5800
8	Trenna	3500	50	ST_CLERK	3500	2925
9	Curtis	3100	50	ST_CLERK	3500	2925

Costo:

Se tienen 5 accesos a la misma tabla y una cantidad importante de operaciones.

Sentencia 2:

```
SELECT
   FIRST_NAME NOMBRE, SALARY SALARIO, DEPARTMENT_ID DEPARTAMENTO,
   JOB_ID CARGO,ROUND(MEDIA_DEPTO,0)PROMEDIO_DEPTO,ROUND(MEDIA_CARGO,0)PROMEDIO_CARGO
FROM
    (SELECT
      FIRST_NAME, SALARY, DEPARTMENT_ID, JOB_ID,
      AVG(SALARY) OVER (PARTITION BY JOB_ID) MEDIA_CARGO,
      AVG(SALARY) OVER (PARTITION BY DEPARTMENT_ID)MEDIA_DEPTO
    FROM EMPLOYEES)EMPL
WHERE EMPL.SALARY > EMPL.MEDIA_CARGO OR EMPL.SALARY > EMPL.MEDIA_DEPTO
ORDER BY 2 DESC;
```

Resultado:

	NOMBRE	SALARIO	DEPARTAMENTO	CARGO	PROMEDIO_DEPTO	PROMEDIO_CARGO
1	Steven	24000	90	AD_PRES	19333	24000
2	Michael	13000	20	MK_MAN	9500	13000
3	Shelley	12000	110	AC_MGR	10150	12000
4	Ellen	11000	80	SA_REP	10033	8867
5	Eleni	10500	80	SA_MAN	10033	10500
6	Alexander	9000	60	IT_PROG	6400	6400
7	Kevin	5800	50	ST_MAN	3500	5800
8	Trenna	3500	50	ST_CLERK	3500	2925
9	Curtis	3100	50	ST_CLERK	3500	2925

Costo:

En este caso se puede evidenciar un solo acceso a la tabla y con un número reducido de operaciones.

Con la demostración anterior, se invita a profundizar en el conocimiento de las funciones propias de Oracle y su continuo y correcto uso.

12.4 Creación de índices.

A diferencia de cómo se pueden ver organizados los resultados al generar una consulta, por default la data insertada en un objeto es contenida en tablas apiladas que no tienen ningún orden en particular, de manera que, para recuperar un registro Oracle debe recorrer todo el segmento de tabla a menos que tenga forma de conocer su ubicación específica.

El principal recurso de optimización es la creación de índices sobre columnas no indexadas, siendo estructuras más versátiles y que permiten mejorar el desempeño para la mayoría de los casos.

A manera muy general (y dentro del alcance de este instructivo) se definirán dos tipos de índices denominados: i) Regulares y ii) Basados en funciones.

Se llamarán índices regulares a aquellos cuya sentencia de creación no requiere instrucciones adicionales.

```
CREATE INDEX GESTIONNEW.AGEN_AGEN_IX ON GESTIONNEW.AGENDA(DIAAGENDA)
INITRANS 20 TABLESPACE GESTION_IDX;
```

Se llamarán índices basados en funciones a aquellos en los que se replica el uso de una función que ha sido incluida en la condición de una consulta para mejorar su rendimiento.

Query: empleados contratados en Febrero sin importar el año.

```
SELECT EMPLOYEE_ID
FROM EMPLOYEES
WHERE TO_CHAR(HIRE_DATE,'MM')='02';
```

- Partiendo del supuesto en el que el campo HIRE_DATE es un tipo DATE, el coste varía según el tipo de índice:

Sin índice:

Con índice regular: **CREATE INDEX EMP_HIRE_DATE_IX ON EMPLOYEES(HIRE_DATE);**

Con índice basado en función:

CREATE INDEX EMP_HIRE_DATE_IX2 ON EMPLOYEES(TO_CHAR(HIRE_DATE,'MM'));

Al comparar las estimaciones del optimizador, se evidencia que la consulta requiere menos tiempo y costo al tomar el índice basado en la función expresada en la condición.

Nota: Se recomienda el uso de este tipo de índice sólo cuando no puede generarse la consulta de otra manera, teniendo en cuenta que para upgrades de Oracle podría no generar los mismos resultados y podría requerir nuevamente su optimización.

Manejo De Indices

a) Los registros no se ordenan por la columna condicionada.
b) Los índices de llaves únicas o primarias son creados automáticamente por el motor Oracle (como parte de la creación del constrainst) y son los de más alta selectividad.
c) Normalmente no se deben crear índices en columnas muy actualizadas o en tablas con permanente inserción y borrado, ya que esto genera trabajo adicional en segmentos de undo y redo. Para esto se puede comparar el procesamiento con o sin índices, para medir los beneficios.
d) Solo indexar columnas de mayor selectividad (que traen menos porcentaje de columnas sobre el total). La selectividad de una columna se puede calcular dividiendo el número de registros entre el número de valores distintos de la columna.

e) Normalmente no se deben crear índices en columnas muy actualizadas o en tablas con permanente inserción y borrado, ya que esto genera trabajo adicional en segmentos de undo y redo. Para esto se puede comparar el procesamiento con o sin índices, para medir los beneficios.
f) Indices Compuestos: Se recomienda crear índices compuestos en columnas que se usan regularmente en where compuestos, la selectividad sobre estas columnas es menor que sobre las columnas en forma individual.

CUANDO NO UTILIZAR INDICES

g) Cuando se hacen operaciones o se aplican funciones sobre las columnas indexadas.
h) En un índice compuesto la primera columna no se tiene en las condiciones del where de la sentencia SQL.
i) Cuendo se tiene un hint obligando a un ACCESS FULL.

12.5 Optimización en el SQL

Para poder crear sentencias optimizadas se recomienda los siguientes pasos:

- Evitar el uso del asterisco (*) en la sentencia SELECT, ya sea en la nominación de columnas o en funciones –COUNT(*) por ejemplo. En su lugar agregar los nombres de las columnas o de la ubicación de la llave primaria de la tabla según sea el caso, por ejemplo
  - SELECT NOMBRE, SALARIO FROM TABLA; --Selección de campos.
  - SELECT COUNT(1) FROM TABLA; --Conteo de registros.

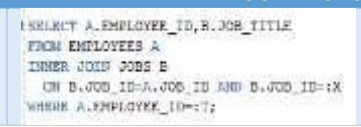
INCORRECTO	CORRECTO

`SELECT *`
`FROM EMPLOYEES`
`WHERE EMPLOYEE_ID = :T;`
	`SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME`
	`FROM EMPLOYEES`
	`WHERE EMPLOYEE_ID = :T;`

`SELECT COUNT(*)`
`FROM EMPLOYEES`
`WHERE EMPLOYEE_ID = :T;`
	`SELECT COUNT(1)`
	`FROM EMPLOYEES`
	`WHERE EMPLOYEE_ID = :T;`

- Utilizar alias para la identificación de las tablas y a su vez los respectivos campos.
- Siempre se debe construir la sentencia de la tabla con mayor cantidad de datos a la menor.
- Para las tablas paramétricas se debe utilizar la restricción left join o rigth join, para que no sean limitantes a la hora de traer registros.
- No realizar sub-consultas en el SELECT pues para un uso óptimo se requieren análisis comparativos de traza, costo y entorno de ejecución. El uso de sub-consultas no siempre optimizan el rendimiento. En su lugar usar JOIN según se requiera.
- En las condiciones del SELECT, clausulas WHERE u ON (depende de la restricción), procurar usar inicialmente los campos con índices y por último los campos no indexados (por características especiales –uso de LIKE, por ejemplo). En la sección del WHERE procurar

que sólo vayan las condiciones de la tabla seguida de FROM y en la cláusula ON las condiciones de correspondientes a la tabla seguida de JOIN.

INCORRECTO	CORRECTO
	

- Las condiciones entre las secciones WHERE u ON, donde se comparen 2 campos deben compartir las mismas características de tipos de campo y validación de nulidad, con el fin que no se requiera de una conversión del datos, siempre y cuando sea posible
- A la hora de utilizar un lenguaje de programación para enviar una instrucción hacia el motor de bases de datos se deben emplear las variables *bind*, para que el motor pueda detectar el mejor método de optimización interno.
- Consultar con el DBA el costo de máquina de la consulta, para determinar si hacen falta índices, mejores relaciones o empleo de HINT, ya sea para disminuir el costo, para aumentar la velocidad de respuesta o un punto medio entre estos dos indicadores.
- Se debe tener cuidado con las agrupaciones, transformaciones en pivote y operaciones, ya que no en todos los casos el motor de la base responde más rápido aunque el aplicativo lo gestione.
- Se pueden utilizar cursores para manejo de datos, sobre todo en el tema de inserciones, actualizaciones o borrado de datos de forma masiva junto con el uso de BULK COLLECT / INSERT ALL, pero se debe controlar la ejecución del commit teniendo en cuenta el total de registros a trabajar; por ejemplo si son 10000 datos a insertar, un commit cada 1000 es una buena práctica.
- Si se realiza un TRUNCATE a una tabla para después inyectar un lote de datos, se recomienda regenerar las estadísticas de la misma:
    - BEGIN DBMS_STATS.GATHER_TABLE_STATS(OWNNAME => 'Nombre del esquema', TABNAME => 'Nombre de la tabla', DEGREE => 4,CASCADE => TRUE); END;

12.6 Uso de Hints.

Un HINT es una instrucción al optimizador. Al escribir SQL el desarrollador/DBA puede conocer información acerca de la data desconocida al optimizador, los hints permiten tomar decisiones que generalmente toma el optimizador, a veces lo fuerza a seleccionar un plan que ve como de mayor costo.

El uso de los hints debe ser administrado, revisado y controlado porque pueden volverse obsoletos o generar mayores costos al aplicar cambios en la base de datos.

Existen diferentes tipos y categorías dependiendo de la cantidad de tablas y el aspecto a optimizar (por el alcance de este instructivo se explicarán los más comunes).

Los hints se agregan en un comentario de tipo /*+ HINT */ precedido de SELECT, UPDATE, INSERT, MERGE o DELETE (Oracle no recomienda su uso con vistas).

- INDEX y NO_INDEX.
  INDEX instruye al optimizador para escanear un índice para la tabla especificada; NO_INDEX evita que el optimizador tenga en cuenta un índice especificado. Puede agregarse _ASC o _DESC a INDEX si se tiene incluido un ORDER BY en la consulta.

```
SELECT /*+ INDEX (employees emp_department_ix)*/ employee_id, department_id
  FROM employees
  WHERE department_id > 50;
```

- PARALLEL
  Permite generar procesos adicionales que llevan a cabo una o más operaciones paralelamente. Puede parametrizarse a nivel de tabla o de sentencia y con diversos tipos de parámetro en cualquier caso, se usará con parámetro numérico y éste no debe ser mayor a 3.

```
SELECT /*+ PARALLEL(2)*/ FIRST_NAME
FROM EMPLOYEES;
```

13. Ejemplos prácticos.

13.1 Creación de tabla particionada.

El particionado de tablas permite un almacenamiento óptimo para data histórica de gran volumen a través de la segmentación lógica de los datos en secciones independientes según determinado criterio (Para mayor detalle por favor consulte la documentación Oracle):

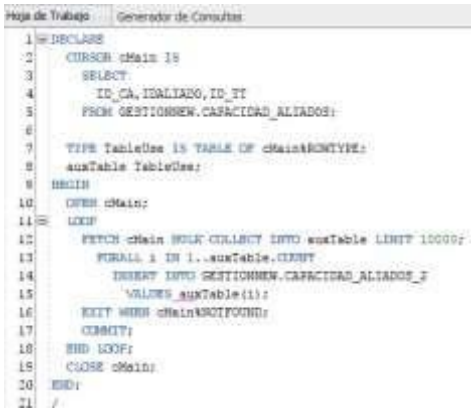
- Range. Intervalos de valores (mayormente usada con columnas tipo fecha).
- Hash. Distribución uniforme de la data entre las particiones.
- List. Listar juntos datos no relacionados en particiones (ej: Lista de estados en una región).

```
1  CREATE TABLE HR.RECU_EMPLEADOS
2  (
3     EMPLOYEE_ID NUMBER(15) CONSTRAINT RECU_EMPL_FK PRIMARY KEY USING INDEX,
4     START_DATE  DATE CONSTRAINT RECU_EMPL_FK2 NOT NULL,
5     END_DATE    DATE DEFAULT TO_DATE('01/01/1900','DD/MM/YYYY'),
6     JOB_ID      NUMBER(6) CONSTRAINT RECU_EMPL_FK3 NOT NULL,
7     DEPARTMENT_ID NUMBER(4) CONSTRAINT RECU_EMPL_FK4 NOT NULL
8  )
9  PARTITION BY RANGE(START_DATE)
10 INTERVAL (NUMTOYMINTERVAL(1,'MONTH'))
11 (
12   PARTITION RECU_PART_PRT1 VALUES LESS THAN (TO_DATE('01/02/2014','DD/MM/YYYY'))INITRANS 20 TABLESPACE TB_DATOS
13 )
14 TABLESPACE TB_DATOS;
```

En el ejemplo se observa un particionado tipo RANGE tomando como argumento el atributo START_DATE que dividirá los registros en particiones mensuales. La instrucción requiere que se incluya como mínimo una línea de partición, a partir de allí se replica la instrucción automáticamente.

13.2 Inserción masiva de datos

Para procesos masivos se sugiere la implementación de un procedimiento almacenado cuya implementación de cursores y ciclos permiten un desempeño más eficiente y eficaz con menor impacto de tiempo y procesamiento.



cMain: Cursor de captura de data desde una tabla origen.

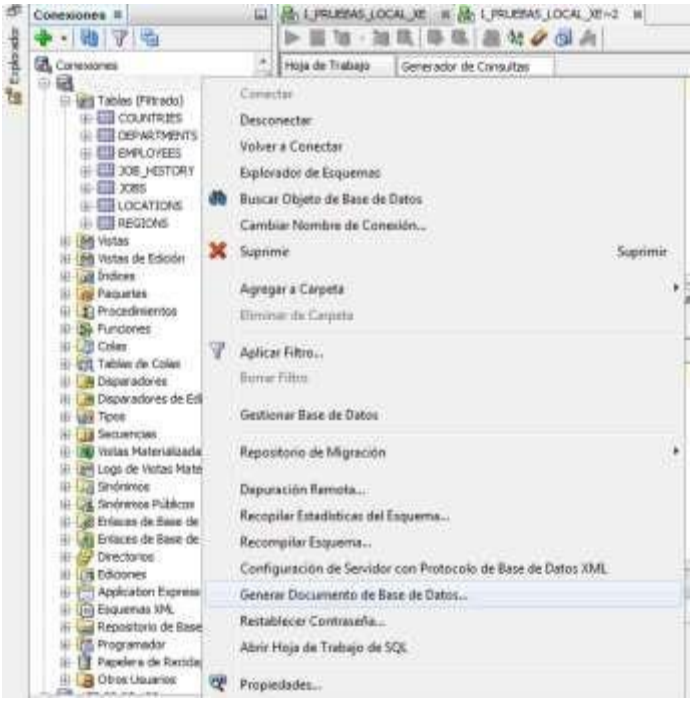
auxTable: Tipo TABLE para almacenamiento de la data capturada por cMain.

BULK COLLECT INTO: Instrucción para depositar la data de cMain en auxTable.

13.3 Export del Data Dictionary

La metadata de la base de datos puede ser exportada para PAP según se indica en el último punto de ese apéndice (más info)

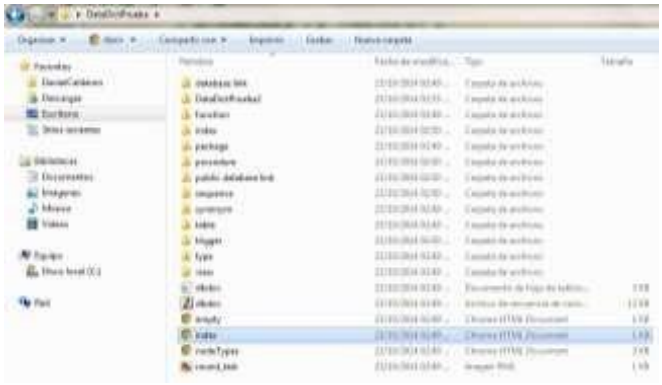
1. Clic derecho en la conexión \ Generar Documento de Base de Datos …



2. Selección de directorio de salida y selección de objetos.

La configuración por default puede conservarse a menos que haya solicitud expresa que indique lo contrario.

3. Se genera finalmente una carpeta con la metadata en HTML.



4. El archivo <index.html> permite la visualización.

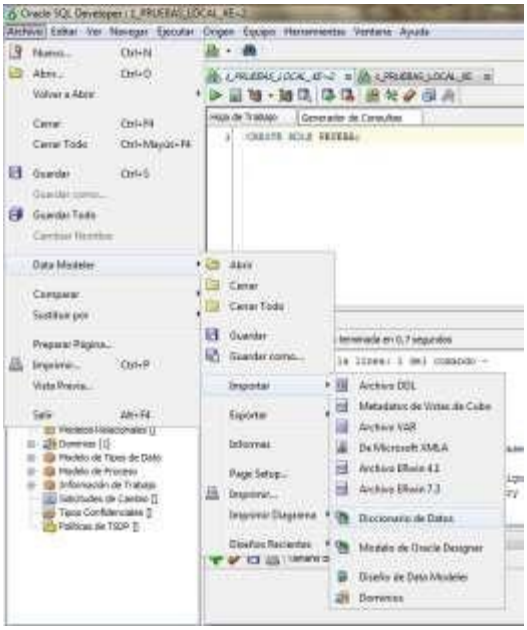
Nota: Es necesario asegurarse que el usuario con el cual se genera la conexión tenga los privilegios suficientes para visualizar todos los objetos necesarios; puede suceder que haya objetos ocultos al usuario.

13.4 Visualización y Export del MER en SQL Developer.

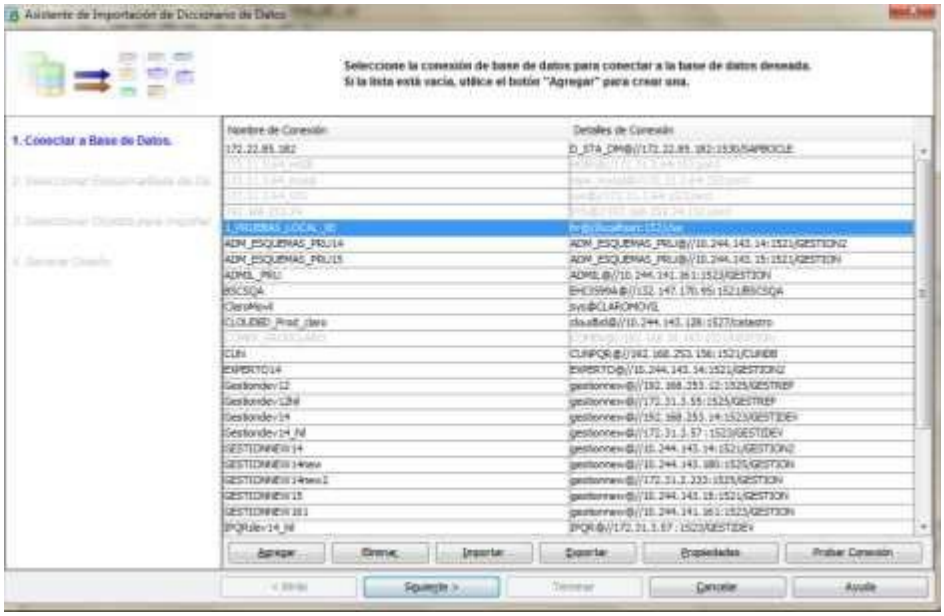
El proceso de exportación del Modelo Entidad Relación sólo requiere los siguientes pasos:

1. Visualización:
    a. Importación del DataDictionary.

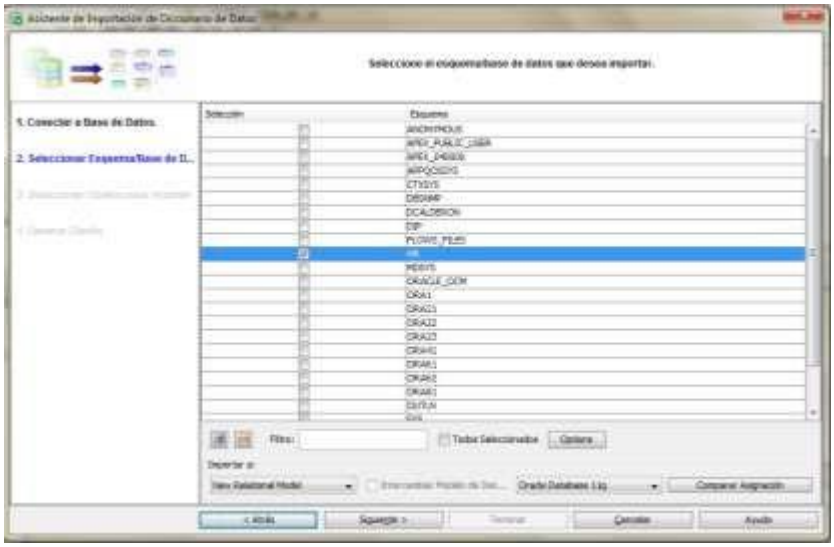
       Archivo\Data Modeler\Importar\Diccionario de Datos.



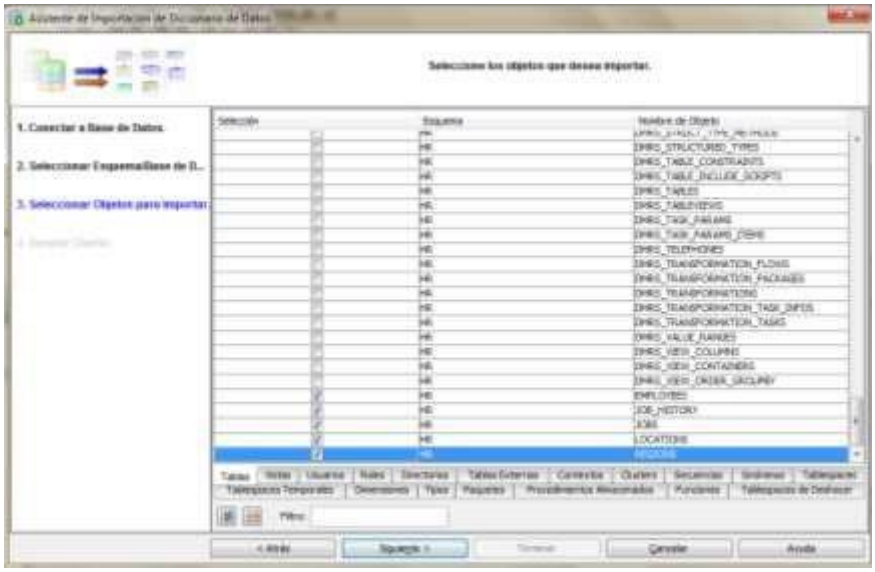
    b. Selección de la conexión (y siguiente):



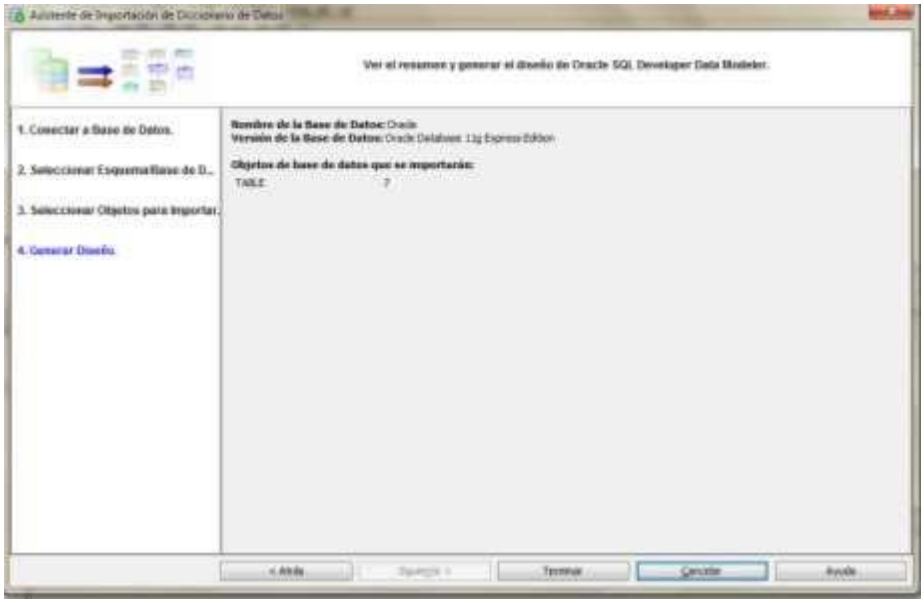
c. Seleccionar Esquema/Base de Datos (puede ser múltiple).



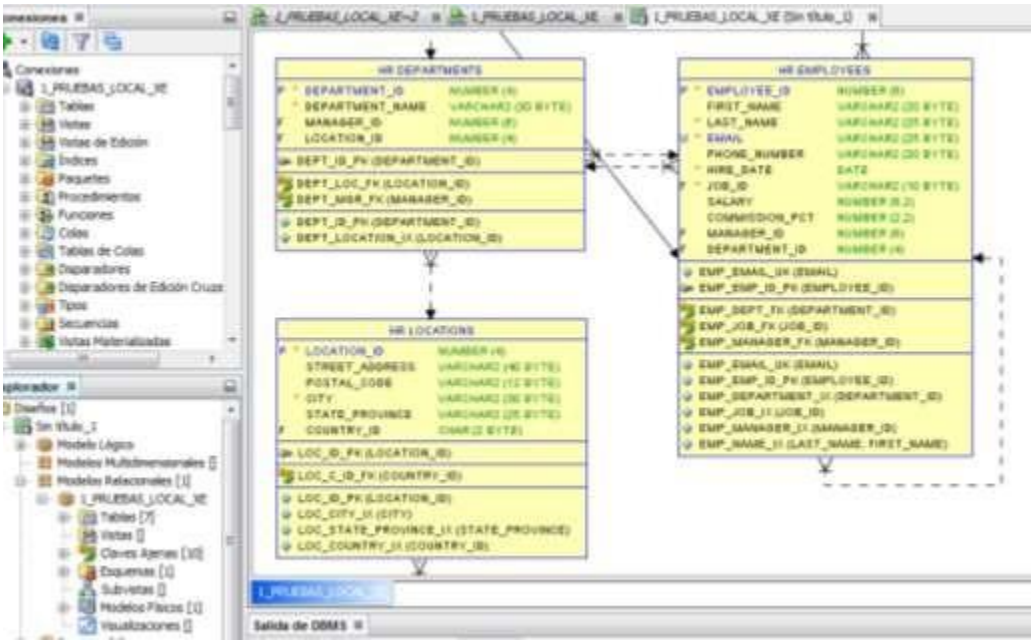
d. Selección de objetos a importar (en este ejemplo sólo se importarán tablas).



e. Confirmar.

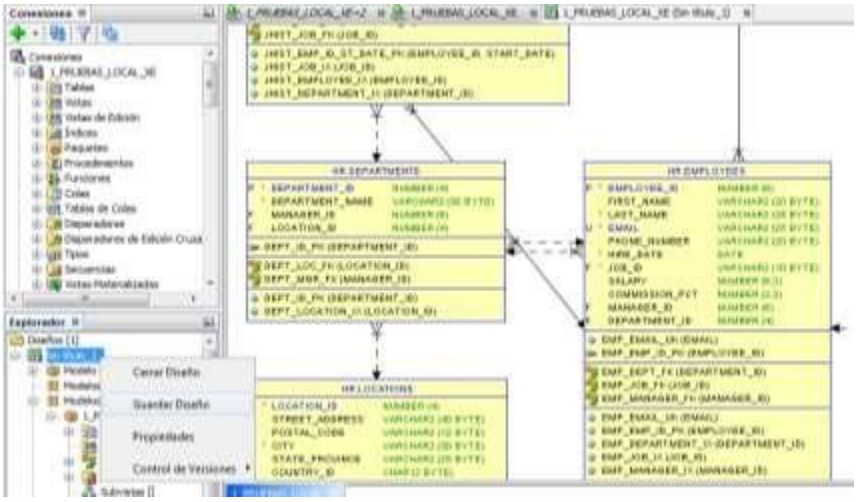


A partir de este proceso puede visualizarse el modelo depositado en el área de explorador de Data Modeler (dentro de SQL Developer).

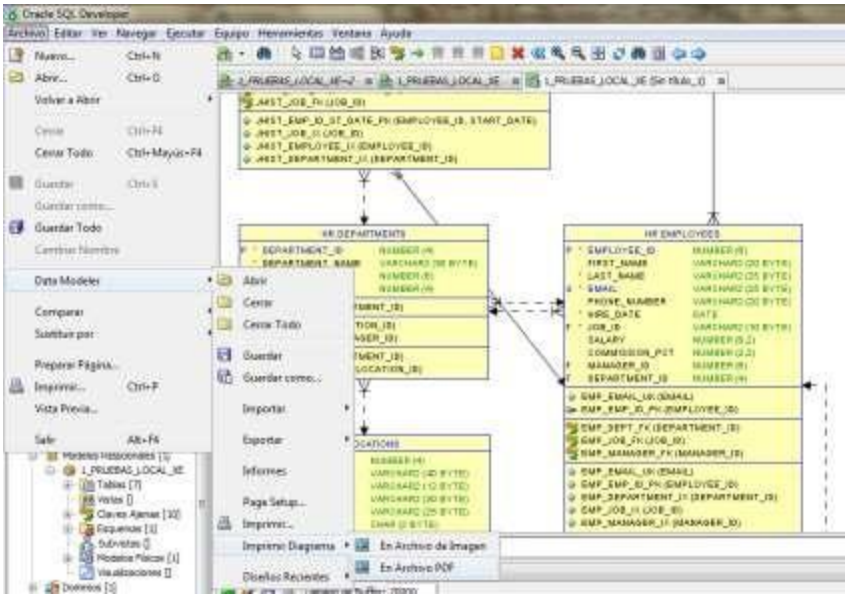


2. Export.
   Para exportar el MER se tienen (entre otras) las siguientes dos formas:

   a. Guardar diseño como tipo Data Modeler (también se puede importar en SQL Developer).



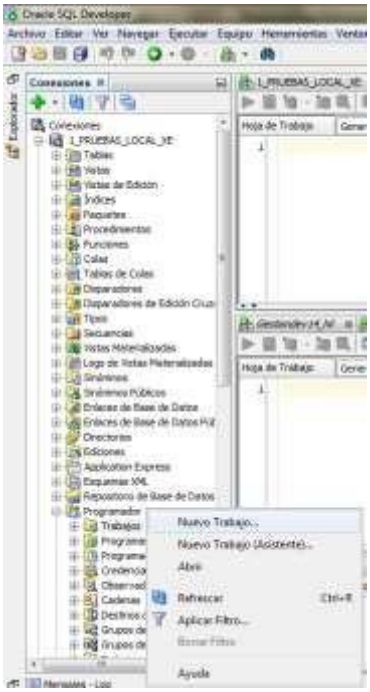
   b. Exportación como imagen o pdf.



13.5 Creación de Jobs.

Un Job se define como la planificación de una tarea en la base de datos con una hora y periodicidad determinadas. Por supuesto, la diferencia entre un Job y una tarea programada de sistema operativo es que el primero se encuentra dentro de la base de datos de manera que si esta no se encuentra funcionando, la tarea no se ejecuta.

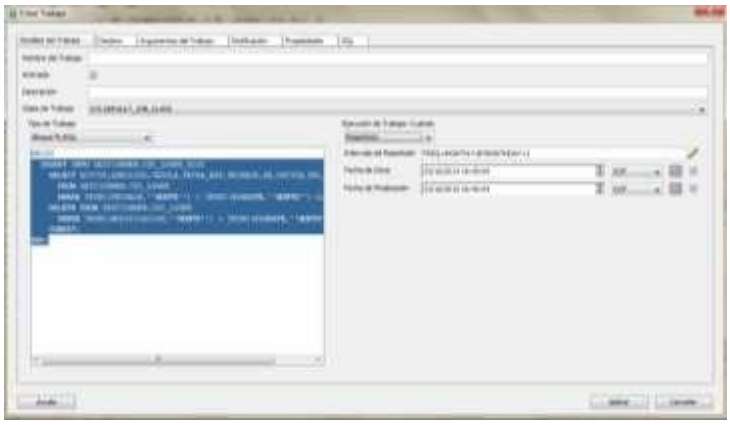
1. Creación del Job:



* Desplegar objetos de la conexión.

* Programador\Trabajos (Clic derecho) \Nuevo trabajo.

* Selección de tipo de trabajo y parámetros. Finalmente, aplicar.



14 Manejo de SPOOL.

Los spool tienen como función generar un log para la trazabilidad en la ejecución de los scripts, permite detectar las fallas en el caso de que la instalación no sea satisfactoria si es el caso.

La apertura se coloca como se ve en la figura antes de la ejecución:

```
SPOOL Objetos.Log

ALTER SESSION SET CURRENT_SCHEMA = MGL_SCHEME;


--------------------------------------------------------
--  DDL for Sequence CMT_AGENDA$AUD_SQ
--------------------------------------------------------

   CREATE SEQUENCE  MGL_SCHEME.CMT_AGENDA$AUD_SQ  MINVALUE 1 MAXVAL
--------------------------------------------------------
--  DDL for Sequence CMT_AGENDA_SQ
--------------------------------------------------------

   CREATE SEQUENCE  MGL_SCHEME.CMT_AGENDA_SQ  MINVALUE 1 MAXVALUE 9
--------------------------------------------------------
--  DDL for Sequence CMT_APROBACIONES_ESTADOS_SQ
--------------------------------------------------------
```

Y el cierre del spool se coloca al final de la ejecución de los scripts:

```
CREATE PUBLIC SYNONYM TEC_TIPO_RED_TEC_HABI$AUD FOR MGL_SCHEME.T
CREATE PUBLIC SYNONYM TEC_TIPO_TECNOLOGIA_HAB FOR MGL_SCHEME.TEC
CREATE PUBLIC SYNONYM TEC_TIPO_TECNOLOGIA_HAB$AUD FOR MGL_SCHEMI
CREATE PUBLIC SYNONYM TEC_VETO FOR MGL_SCHEME.TEC_VETO;
CREATE PUBLIC SYNONYM TEC_VETO_CANAL FOR MGL_SCHEME.TEC_VETO_CAN
CREATE PUBLIC SYNONYM TEC_VETO_CIUDAD FOR MGL_SCHEME.TEC_VETO_CI
CREATE PUBLIC SYNONYM TEC_VETO_NODO FOR MGL_SCHEME.TEC_VETO_NODO


SPOOL OFF
```

Para la definición de los SPOOL se tomaran en cuenta los nombre de los scripts. Ejemplo:

1.Objetos.sql → Objetos.log

2.Desasher.sql → Desasher.log