Joel Mendez – 202317429

Miguel Ibañez - 202325628

Santiago Muñoz - 202221167

Taller 2: Informe laboratorio

1.Analisis creación de tablas

```
CREATE TABLE ARTISTAS (
    ID NUMBER PRIMARY KEY,
    NOMBRE VARCHAR2(120 BYTE)
);

CREATE TABLE ALBUMES (
    ID NUMBER PRIMARY KEY,
    TITULO VARCHAR2(160 BYTE) NOT NULL,
    ARTISTA_ID NUMBER NOT NULL
);
```

Para el desarrollo de cada tabla lo que hicimos fue usar la función CREATE TABLE, esto crea una tabla en sql con el nombre que nosotros queramos, en este caso creamos las respectivas tablas que nos pedían en el modelo de entidad – relación. Creamos la tabla para artistas, álbumes, canciones, géneros, tipos_medio, listas_de_reproduccion y canciones_en_listas.

Adentro de cada función podemos ver cómo están sus respectivos atributos, los cuales se convertirán en columnas, guiándonos del modelo dado.

Para los nombres o textos, ponemos VARCHAR2 y se calcularon considerando la longitud máxima esperada de la información

Asignamos una PRIMARY KEY en cada entidad, dependiendo de su necesidad

```
CREATE TABLE CANCIONES (
    ID NUMBER PRIMARY KEY,
    NOMBRE VARCHAR2(200 BYTE) NOT NULL,
    COMPOSITOR VARCHAR2(220 BYTE),
    MILISEGUNDOS NUMBER NOT NULL,
    BYTES NUMBER,
    PRECIO_UNITARIO NUMBER(10,2) NOT NULL,
    ALBUM_ID NUMBER NOT NULL,
    MEDIO_ID NUMBER NOT NULL,
    GENERO_ID NUMBER NOT NULL
);
```

En esta parte, para los atributos que son numéricos como milisegundos, bytes, precio unitario,etc. Se declararon como NUMBER

NUMBER (10,2) puede guardar datos como: 12345678.90 ya que tiene en total 8 dígitos antes y 2 después sumando un total de 10, y también cumple que siempre habrá hasta 2 decimales.

2.relaciones tablas

```
ALTER TABLE CANCIONES
ADD CONSTRAINT CANCIONES_GENEROS_FK
    FOREIGN KEY (GENERO_ID)
    REFERENCES GENEROS (ID)
    ENABLE;

ALTER TABLE CANCIONES
ADD CONSTRAINT CANCIONES_ALBUMES_FK
    FOREIGN KEY (ALBUM_ID)
    REFERENCES ALBUMES (ID)
    ENABLE;

ALTER TABLE CANCIONES
ADD CONSTRAINT CANCIONES_MEDIOS_FK
    FOREIGN KEY (MEDIO_ID)
    REFERENCES TIPOS_MEDIO (ID)
    ENABLE;
```

En la parte de las relaciones, lo que hicimos fue usar claves foráneas para poder conectar las tablas entre sí, tal como se veía en el diagrama.

Por ejemplo: en la tabla CANCIONES agregamos las llaves GENERO_ID, ALBUM_ID y MEDIO_ID para que cada canción quede relacionada con su género, su álbum y su tipo de medio.

Para esto usamos ALTER TABLE, ADD_CONTRAINT, FOREIGN KEY, REFERENCES, esas columnas van a apuntar a la llave primaria de la otra tabla

Como, por ejemplo: GENERO_ID apunta a GENEROS(ID)

```
CREATE TABLE CANCIONES_EN_LISTA (
    LISTA_ID NUMBER,
    CANCION_ID NUMBER,
    CONSTRAINT CANCIONES_EN_LISTA_PK PRIMARY KEY (LISTA_ID, CANCION_ID)
);

ALTER TABLE CANCIONES_EN_LISTA
ADD CONSTRAINT CANCIONES_EN_LISTA_CANCIONES
    FOREIGN KEY (CANCION_ID)
    REFERENCES CANCIONES (ID)
    ENABLE;

ALTER TABLE CANCIONES_EN_LISTA
ADD CONSTRAINT CANCIONES_EN_LISTA_LISTAS_DE_REPRODUCCION
    FOREIGN KEY (LISTA_ID)
    REFERENCES LISTAS_DE_REPRODUCCION (ID)
    ENABLE;
```

En el caso de la relación muchos a muchos entre Canciones y Listas de reproducción, se creó una tabla intermedia llamada CANCIONES_EN_LISTA.
Ahí se pusieron las columnas LISTA_ID y CANCION_ID, y juntas forman una clave primaria compuesta, esto es para que no se repita la misma canción dos veces en la misma lista.
Después, con los ALTER TABLE, se agregaron las llaves foráneas que conectan con CANCIONES e LISTAS_DE_REPRODUCCION.

3.Consultas

```
-- Primera consulta
SELECT  LISTAS_DE_REPRODUCCION.ID, LISTAS_DE_REPRODUCCION.NOMBRE, NVL(SUM(CANCIONES.PRECIO_UNITARIO), 0) AS PREC
FROM LISTAS_DE_REPRODUCCION
LEFT JOIN CANCIONES_EN_LISTA
        ON CANCIONES_EN_LISTA.LISTA_ID = LISTAS_DE_REPRODUCCION.ID
LEFT JOIN CANCIONES
        ON CANCIONES.ID = CANCIONES_EN_LISTA.CANCION_ID
GROUP BY LISTAS_DE_REPRODUCCION.ID, LISTAS_DE_REPRODUCCION.NOMBRE
ORDER BY PRECIO_TOTAL DESC
FETCH FIRST 3 ROWS ONLY;
```

La idea de esta consulta fue obtener cuáles eran las listas de reproducción con mayor valor según el precio de las canciones que contienen.

Para eso unimos las listas con las canciones por medio de la tabla intermedia y usamos SUM para acumular los precios. También pusimos NVL para que las listas vacías no quedaran en NULL sino en 0. Al final agrupamos por cada lista y ordenamos de mayor a menor precio, limitando el resultado a las 3 primeras.

```sql
--Segunda consulta
SELECT CANCIONES.*
FROM CANCIONES
LEFT JOIN CANCIONES_EN_LISTA
    ON CANCIONES_EN_LISTA.CANCION_ID = CANCIONES.ID
JOIN TIPOS_MEDIO
    ON TIPOS_MEDIO.ID = CANCIONES.MEDIO_ID
WHERE CANCIONES_EN_LISTA.LISTA_ID IS NULL AND TIPOS_MEDIO.NOMBRE = 'AAC audio file';
```

Para esta consulta lo que necesitábamos era encontrar canciones que no estuvieran en ninguna lista de reproducción

Para eso usamos un LEFT JOIN con la tabla intermedia y luego filtramos con IS NULL. Además, quisimos que fueran solo las canciones que tuvieran como tipo de medio “AAC audio file”, en el WHERE se hizo otro JOIN con la tabla de tipos de medio y se agregó esa condición.

```sql
--Tercera consulta

SELECT ARTISTAS.NOMBRE, COUNT(DISTINCT ALBUMES.ID) AS cantidad_albumes, AVG(CANCIONES.PRECIO_UNITARIO)
FROM ARTISTAS
LEFT JOIN ALBUMES
    ON ALBUMES.ARTISTA_ID = ARTISTAS.ID
LEFT JOIN CANCIONES
    ON CANCIONES.ALBUM_ID = ALBUMES.ID
GROUP BY ARTISTAS.NOMBRE
HAVING COUNT( DISTINCT ALBUMES.ID)>=5
ORDER BY COUNT(DISTINCT ALBUMES.ID) DESC, AVG(CANCIONES.PRECIO_UNITARIO) DESC;
```

Para la tercera consulta se quería conocer los nombres de los artistas con más de 5 álbumes registrados, junto con el número total de álbumes y el precio promedio de sus canciones

La estrategia fue unir artistas con sus álbumes y luego con sus canciones, para después agrupar todo por artista. Con COUNT(DISTINCT) contamos los álbumes sin repetir, y con AVG obtuvimos el promedio del precio unitario de las canciones. Luego, con HAVING, dejamos solo a los que tenían 5 o más álbumes y ordenamos primero por la cantidad de álbumes y después por el promedio más alto.