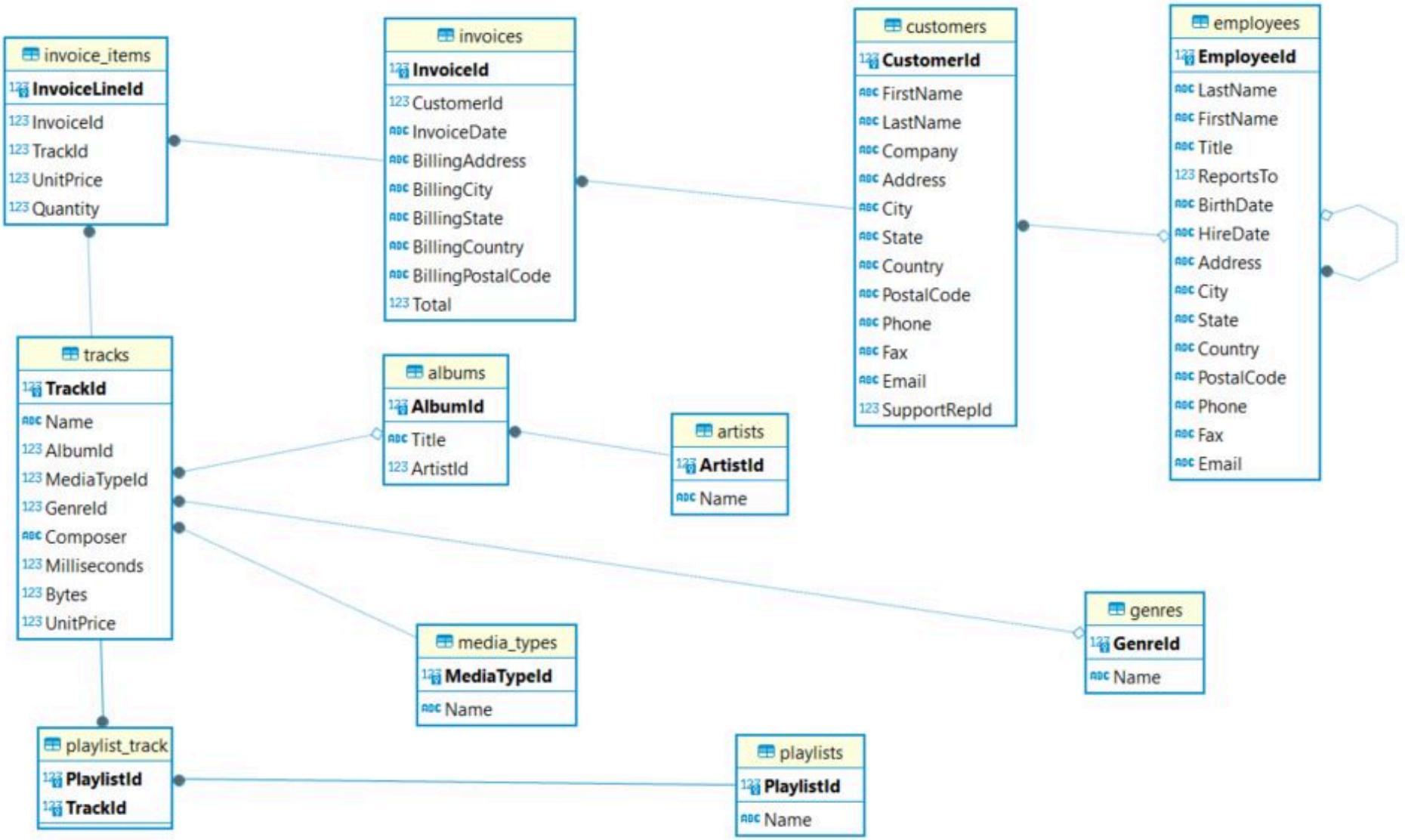
Aan de slag met Python en relationele databanken

>

> Relationele databanken zijn, ruim 40 jaar nog hun introductie, nog steeds veruit de belangrijkste opslagmethode voor gestructureerde gegevens. Deze gegevens dienen niet alleen voor de ondersteuning van de dagelijkse operaties van een organisatie, maar worden ook gebruikt voor geavanceerde rapporteringen en analyses.
>
> Na een korte introductie in relationele databanken en de SQL-taal, ga je in deze workshop zelf aan de slag om vanuit Python een relationele databank aan te spreken. Verschillende gangbare technieken, telkens met hun voor- en nadelen, komen aan bod.
>
> Speciale aandacht gaat hierbij naar de onderhoudbaarheid van software en het omgaan met realistische datavolumes.
>
> We gebruiken hier SQLite als voorbeeld van een relationele databank wegens zijn eenvoud: single-user, single-file, geen server-software nodig. Zie https://db-engines.com/en/ranking voor een overzicht van alle bestaande databasemanagementsystemen.
>
> ir. Johan Decorte
> johan.decorte@hogent.be

Voorbeeld relationele database: Chinook



Voorbeeldstatements

1. **Eenvoudige SELECT**

   Toon trackid, name en duur (in seconden) voor de 5 langste tracks. Sorteer aflopend op duur.

```
select trackid, name, milliseconds/1000
 from tracks
 order by milliseconds desc
 limit 5;
```

2. **Complexere SELECT**

   Wie zijn de 10 best verkopende artisten? Toon ook hun omzet.

```sql
select ar.Name,sum(ii.Quantity*ii.UnitPrice)
 from tracks t join albums al
 on t.AlbumId = al.AlbumId
 join artists ar on al.ArtistId = ar.ArtistId
 join invoice_items ii  on t.TrackId = ii.TrackId
 group by ar.Name
 order by sum(ii.Quantity*ii.UnitPrice) desc
 limit 10;
```

3. **UPDATE**

   Verhoog de prijzen van alle tracks van het genre x met y % (x en y zijn parameters).

```sql
update tracks
 set unitprice = unitprice * (1 + 0.10)
 where GenreId in (select genreid from genres where name = 'Pop');
```

4. **INSERT**

   Voeg de nieuwe artiest "Camille" toe.

```sql
-- veld ArtistId is autonumber en wordt dus automatisch toegekend als volgende vrije getal.
INSERT INTO artists(Name) VALUES('Camille')
```

5. **DELETE**

   Verwijder artiest "Camille".

```sql
DELETE FROM artists WHERE name='Camille'
```

INHOUD

1. Plain SQL in Python code

- SQL is voor Python slechts een string
- Gebruik een database-specifieke Python-libary, bijv. sqlite3 voor SQLite

1.1. Initiële setup

In [ ]:
```
#####################################
# Plain SQL: native SQLite API
#####################################
print ('*** Native SQLite API ***')

import sqlite3

# Connect to the database
conn = sqlite3.connect(r"C:\sqlite\db\chinook.db")
cur = conn.cursor()
```

```
*** Native SQLite API ***
```

1.2. Statement 1

```
select trackid, name, milliseconds/1000
from tracks
order by milliseconds desc
limit 5;
```

In [ ]:
```
stmt = "select trackid, name, milliseconds/1000 from tracks order by milliseconds desc limit 5"
cur.execute(stmt)
rows = cur.fetchall()
print(f'Rows is of data type {type(rows)}')
print()

print(f'{"ID":<5} {"Name":<30} {"Seconds":>7}')
print('-'*44)
for row in rows:
    print(f'{row[0]:<5} {row[1]:<30} {round(row[2]):>7}')
```

```
Rows is of data type <class 'list'>

ID    Name                           Seconds
--------------------------------------------
2820  Occupation / Precipice            5286
3224  Through a Looking Glass           5088
3244  Greetings from Earth, Pt. 1       2960
3242  The Man With Nine Lives           2956
3227  Battlestar Galactica, Pt. 2       2956
```

1.3. Statement 2

```
select ar.Name,sum(ii.Quantity*ii.UnitPrice)
from tracks t join albums al
on t.AlbumId = al.AlbumId
join artists ar on al.ArtistId = ar.ArtistId
join invoice_items ii  on t.TrackId = ii.TrackId
group by ar.Name
order by sum(ii.Quantity*ii.UnitPrice) desc
limit 10;
```

In [ ]:
```
stmt = """
        select ar.Name,sum(ii.Quantity*ii.UnitPrice)
        from tracks t join albums al
        on t.AlbumId = al.AlbumId
        join artists ar on al.ArtistId = ar.ArtistId
        join invoice_items ii  on t.TrackId = ii.TrackId
        group by ar.Name
        order by sum(ii.Quantity*ii.UnitPrice) desc
        limit 10;
        """

cur.execute(stmt)
rows = cur.fetchall()

print(f'{"Name":<30} {"Sales":>7}')
print('-'*38)
for row in rows:
    print(f'{row[0]:<30} {round(row[1],2):7.2f}' )
```

```
Name                           Sales
--------------------------------------
Iron Maiden                   138.60
U2                            105.93
Metallica                      90.09
Led Zeppelin                   86.13
Lost                           81.59
The Office                     49.75
Os Paralamas Do Sucesso        44.55
Deep Purple                    43.56
Faith No More                  41.58
Eric Clapton                   39.60
```

1.4. Statement 3

```
update tracks
set unitprice = unitprice * (1 + ?/100.0)
where GenreId in (select genreid from genres where name = ?);
```

In [ ]:
```
# Statement 3

stmt = """
    update tracks
    set unitprice = unitprice * (1 + ?/100.0)
    where GenreId in (select genreid from genres where name = ?);
    """
cur.execute(stmt, (10, 'Pop') )
```

Out[ ]:
```
<sqlite3.Cursor at 0x2c593d4fc40>
```

In [ ]:
```
# Check results
stmt = "select trackid, unitprice from tracks where GenreId in (select genreid from genres where name = 'Pop')"
cur.execute(stmt)
rows = cur.fetchall()

print(f'{"ID":<30} {"Unitprice":>7}')
print('-'*38)
for row in rows:
    print(f'{row[0]:<30} {round(row[1],2):7.2f}' )
```

ID	Unitprice
323	1.09
324	1.09
325	1.09
326	1.09
327	1.09
328	1.09
329	1.09
330	1.09
331	1.09
332	1.09
333	1.09
334	1.09
335	1.09
336	1.09
3253	1.09
3254	1.09
3255	1.09
3256	1.09
3257	1.09
3258	1.09
3259	1.09
3260	1.09
3261	1.09
3262	1.09
3263	1.09
3264	1.09
3265	1.09
3266	1.09
3267	1.09
3268	1.09
3269	1.09
3270	1.09
3271	1.09
3272	1.09
3273	1.09
3274	1.09
3275	1.09
3467	1.09
3468	1.09
3469	1.09
3470	1.09
3471	1.09
3472	1.09
3473	1.09
3474	1.09
3475	1.09

```
3476                            1.09
3477                            1.09
```

Ofwel, updates terugdraaien...

In [ ]:
```
conn.rollback()
```

...ofwel, updates bevestigen.

In [ ]:
```
conn.commit()
```

1.5 Statement 4

```
INSERT INTO artists(Name) VALUES('Camille')
```

In [ ]:
```
stmt = "INSERT INTO artists(Name) VALUES('Camille');"
cur.execute(stmt )
```

Out[ ]:
```
<sqlite3.Cursor at 0x2c593d4fc40>
```

In [ ]:
```
# Check results
stmt = "SELECT * FROM artists ORDER BY artistid DESC LIMIT 1;"
cur.execute(stmt)
rows = cur.fetchall()

print(f'{"ArtistId":<10} {"Name":<30}')
print('-'*32)
for row in rows:
    print(f'{row[0]:<10} {row[1]:<30}' )
```

```
ArtistId   Name
--------------------------------
284        Camille
```

Voer `rollback()` uit om de wijzigingen terug te draaien, of `commit()` om ze definitief te maken.

In [ ]:
```
conn.rollback()
```

In [ ]:
```
conn.commit()
```

1.6 Statement 5

```
DELETE FROM artists WHERE name='Camille'
```

In [ ]:
```
stmt = "DELETE FROM artists WHERE name='Camille';"
cur.execute(stmt )
```

Out[ ]:
```
<sqlite3.Cursor at 0x2c593d4fc40>
```

In [ ]:
```
# Check results
stmt = "SELECT * FROM artists ORDER BY artistid DESC LIMIT 1;"
cur.execute(stmt)
rows = cur.fetchall()

print(f'{"ArtistId":<10} {"Name":<30}')
print('-'*32)
for row in rows:
    print(f'{row[0]:<10} {row[1]:<30}' )
```

```
ArtistId   Name
--------------------------------
275        Philip Glass Ensemble
```

In [ ]:
```
conn.rollback()
```

In [ ]:
```
conn.commit()
```

1.7. Afsluiten

In [ ]:
```
conn.close()
```

1.8. Voor- en nadelen van "Plain SQL"

- `+` Eenvoudig in gebruik: test je SQL string m.b.v. een SQL tool (bijv. DBeaver) en copy-paste naar Python
- `-` Geen SQL syntax check bij de ontwikkeling --> errors duiken op at runtime
- `-` Geen integratie met Python objecten.
- `-` Code is niet porteerbaar naar andere databanken:
  - `-` Je gebruikt software-bibliotheken die specifiek zijn voor een bepaald database-systeem (= native API)
  - `-` SQL-code volgt dialect van een bepaalde database en is dus niet noodzakelijk porteerbaar naar een andere database.

2. Language Integrated Query (LINQ)

- Bouw het SQL-commando op aan de hand van Python-functies.

- Gebruik een database-onafhankelijke Python-library: SQL Alchemy Core.
- Tabellen als geheel zijn objecten.

2.1. Initiële setup

In [ ]:
```
###############################################################################
#  LINQ (Language Integrated Query) met SQL Alchemy Core -->
###############################################################################
import sqlalchemy
sqlalchemy.__version__
```

Out[ ]:
```
'2.0.29'
```

In [ ]:
```
# Establishing a connection
print ('*** SQL Alchemy Core ***')

from sqlalchemy import create_engine
engine = create_engine(r"sqlite:///C:\sqlite\db\chinook.db")

conn = engine.connect()
```

```
*** SQL Alchemy Core ***
```

In [ ]:
```
# Setting up MetaData with Table Objects
from sqlalchemy import MetaData, Table, Column, Integer, String, Float
metadata = MetaData()   # This object is essentially a facade around a Python dictionary
                        # that stores a series
                        # of Table objects keyed to their string name.

# Option 1: explicit Table objects
Tracks = Table('tracks', metadata,
               Column('TrackId', Integer, primary_key=True),
               Column('Name', String),
               Column('AlbumId', Integer),
               Column('MediaTypeId', Integer),
               Column('GenreId', Integer),
               Column('Composer', String),
               Column('Milliseconds', Integer),
               Column('Bytes', Integer),
               Column('UnitPrice', Float))

# Option 2: refecting tables: generate Table objects automatically from database
Albums = Table('albums', metadata, autoload_with=engine)
Artists = Table('artists', metadata, autoload_with=engine)
Genres = Table('genres', metadata, autoload_with=engine)
InvoiceItems = Table('invoice_items', metadata, autoload_with=engine)
```

2.2. Statement 1

```
select trackid, name, milliseconds/1000
from tracks
order by milliseconds desc
limit 5;
```

In [ ]:
```
from sqlalchemy import select
stmt = select(Tracks.c.TrackId,Tracks.c.Name, Tracks.c.Milliseconds/1000) \
       .order_by(Tracks.c.Milliseconds.desc()) \
       .limit(5)

rows = conn.execute(stmt)

print(f'{"ID":<5} {"Name":<30} {"Seconds":>7}')
print('-'*44)
for row in rows:
    print(f'{row[0]:<5} {row[1]:<30} {round(row[2]):>7}')
```

```
ID    Name                              Seconds
-----------------------------------------------
2820  Occupation / Precipice              5287
3224  Through a Looking Glass             5089
3244  Greetings from Earth, Pt. 1         2960
3242  The Man With Nine Lives             2957
3227  Battlestar Galactica, Pt. 2         2956
```

2.3. Statement 2

```
select ar.Name,sum(ii.Quantity*ii.UnitPrice)
from tracks t join albums al
on t.AlbumId = al.AlbumId
join artists ar on al.ArtistId = ar.ArtistId
join invoice_items ii  on t.TrackId = ii.TrackId
group by ar.Name
order by sum(ii.Quantity*ii.UnitPrice) desc
limit 10;
```

In [ ]:

```
from sqlalchemy import select, func

stmt = select(Artists.c.Name,func.sum(InvoiceItems.c.Quantity * InvoiceItems.c.UnitPrice)) \
       .select_from(Tracks) \
       .join(Albums, Tracks.c.AlbumId == Albums.c.AlbumId) \
       .join(InvoiceItems, Tracks.c.TrackId == InvoiceItems.c.TrackId) \
       .join(Artists, Albums.c.ArtistId == Artists.c.ArtistId) \
       .group_by(Artists.c.Name) \
       .order_by(func.sum(InvoiceItems.c.Quantity * InvoiceItems.c.UnitPrice).desc()) \
       .limit(10)

rows = conn.execute(stmt)

print(f'{"Name":<30} {"Sales":>7}')
print('-'*38)
for row in rows:
    print(f'{row[0]:<30} {round(row[1],2):7.2f}' )
```

```
Name                                Sales
-----------------------------------------
Iron Maiden                        138.60
U2                                 105.93
Metallica                           90.09
Led Zeppelin                        86.13
Lost                                81.59
The Office                          49.75
Os Paralamas Do Sucesso             44.55
Deep Purple                         43.56
Faith No More                       41.58
Eric Clapton                        39.60
```

2.4. Statement 3

```
update tracks
set unitprice = unitprice * (1 + ?/100.0)
where GenreId in (select genreid from genres where name = ?);
```

In [ ]:
```
from sqlalchemy import update,bindparam

subq = select(Genres.c.GenreId).where(Genres.c.Name == bindparam("genre"))

stmt = (update(Tracks).values(UnitPrice=Tracks.c.UnitPrice * (1 + bindparam("pct"))).where(Tracks.c.GenreId.in_(subq)))
print(stmt) # string representation of the statement
conn.execute(stmt,{"pct":0.1, "genre":"Pop"})
```

Out[ ]:
```
UPDATE tracks SET "UnitPrice"=(tracks."UnitPrice" * (:param_1 + :pct)) WHERE tracks."GenreId" IN (SELECT genres."GenreId"
FROM genres
WHERE genres."Name" = :genre)
<sqlalchemy.engine.cursor.CursorResult at 0x2c5946a5fd0>
```

In [ ]:
```
# Check results
from sqlalchemy import select

subq = select(Genres.c.GenreId).where(Genres.c.Name == 'Pop')

stmt = select(Tracks.c.TrackId,Tracks.c.Name, Tracks.c.UnitPrice) \
        .where(Tracks.c.GenreId.in_(subq))
rows = conn.execute(stmt)

print(f'{"ID":<5} {"Name":<40} {"Unitprice":>7}')
print('-'*54)
for row in rows:
    print(f'{row[0]:<5} {row[1]:<40} {row[2]:7.2f}')
```

ID	Name	Unitprice
323	Dig-Dig, Lambe-Lambe (Ao Vivo)	1.09
324	Pererê	1.09
325	TriboTchan	1.09
326	Tapa Aqui, Descobre Ali	1.09
327	Daniela	1.09
328	Bate Lata	1.09
329	Garotas do Brasil	1.09
330	Levada do Amor (Ailoviu)	1.09
331	Lavadeira	1.09
332	Reboladeira	1.09
333	É que Nessa Encarnação Eu Nasci Manga	1.09
334	Reggae Tchan	1.09
335	My Love	1.09
336	Latinha de Cerveja	1.09
3253	Instant Karma	1.09
3254	#9 Dream	1.09
3255	Mother	1.09
3256	Give Peace a Chance	1.09
3257	Cold Turkey	1.09
3258	Whatever Gets You Thru the Night	1.09
3259	I'm Losing You	1.09
3260	Gimme Some Truth	1.09
3261	Oh, My Love	1.09
3262	Imagine	1.09
3263	Nobody Told Me	1.09
3264	Jealous Guy	1.09
3265	Working Class Hero	1.09
3266	Power to the People	1.09
3267	Imagine	1.09
3268	Beautiful Boy	1.09
3269	Isolation	1.09
3270	Watching the Wheels	1.09
3271	Grow Old With Me	1.09
3272	Gimme Some Truth	1.09
3273	[Just Like] Starting Over	1.09
3274	God	1.09
3275	Real Love	1.09
3467	Intro / Stronger Than Me	1.09
3468	You Sent Me Flying / Cherry	1.09
3469	F**k Me Pumps	1.09
3470	I Heard Love Is Blind	1.09
3471	(There Is) No Greater Love (Teo Licks)	1.09
3472	In My Bed	1.09
3473	Take the Box	1.09
3474	October Song	1.09
3475	What Is It About Men	1.09

```
3476  Help Yourself                    1.09
3477  Amy Amy Amy (Outro)              1.09
```

In [ ]:
```
conn.rollback()
```

In [ ]:
```
conn.commit()
```

2.5 Statement 4

```
INSERT INTO artists('Name') VALUES('Camille')
```

In [ ]:
```
from sqlalchemy import insert
stmt = insert(Artists).values(Name='Camille')
conn.execute(stmt)
```

Out[ ]:
```
<sqlalchemy.engine.cursor.CursorResult at 0x2c5946a54e0>
```

In [ ]:
```
# Check results
from sqlalchemy import select

stmt = select(Artists).order_by(Artists.c.ArtistId.desc()).limit(1)
rows = conn.execute(stmt)

print(f'{"ArtistId":<10} {"Name":<30}')
print('-'*32)
for row in rows:
    print(f'{row[0]:<10} {row[1]:<30}' )
```

```
ArtistId   Name
--------------------------------
284        Camille
```

In [ ]:
```
conn.rollback()
```

In [ ]:
```
conn.commit()
```

2.6 Statement 5

```
DELETE FROM artists WHERE name='Camille'
```

In [ ]:
```
from sqlalchemy import delete
stmt = delete(Artists).where(Artists.c.Name == 'Camille')
conn.execute(stmt)
```

Out[ ]: <sqlalchemy.engine.cursor.CursorResult at 0x2c5946a6ac0>

In [ ]:
```
# Check results
from sqlalchemy import select

stmt = select(Artists).order_by(Artists.c.ArtistId.desc()).limit(1)
rows = conn.execute(stmt)

print(f'{"ArtistId":<10} {"Name":<30}')
print('-'*32)
for row in rows:
    print(f'{row[0]:<10} {row[1]:<30}' )
```

```
ArtistId   Name
--------------------------------
275        Philip Glass Ensemble
```

In [ ]:
```
conn.rollback()
```

In [ ]:
```
conn.commit()
```

2.7. Afsluiten

In [ ]:
```
conn.close()
```

2.8. Voor- en nadelen van LINQ

- `+` SQL syntax check door Python --> fouten ontdekt bij het ontwikkelen
- `+` Porteerbaar tussen databanksystemen
- `-` Geen integratie met Python-objecten.
- `-` Extra syntax moet aangeleerd worden.

3. Object Relational Mapping (ORM)

- Werk volledig op basis van Python-objecten.
- Gebruik een database-onafhankelijke Python-library: SQL Alchemy ORM

3.1. Initiële setup

In [ ]:
```
###############################################################################################
# Database independant + application objects --> SQL Alchemy ORM
###############################################################################################

# Establishing a connection
print ('*** SQL Alchemy ORM ***')

from sqlalchemy import create_engine
from sqlalchemy.orm import Session

engine = create_engine(r"sqlite:///C:\sqlite\db\chinook.db")

conn = engine.connect()

session = Session(engine)
```

```
*** SQL Alchemy ORM ***
```

Hier definiëren we constructies op moduleniveau die de structuren zullen vormen die we zullen bevragen vanuit de database.
Deze structuur, bekend als een Declarative Mapping, definieert zowel een Python objectmodel,
als database metadata die echte SQL tabellen beschrijft die bestaan, of zullen bestaan, in een bepaalde database.

In [ ]:
```
from sqlalchemy import String, select
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column
from sqlalchemy import MetaData, Table, Column, Integer, String, Float

class Base(DeclarativeBase):
    pass

# Option 1: explicit Table objects

class Tracks(Base):
    __tablename__ = "tracks"
    TrackId: Mapped[int] = mapped_column(primary_key=True)
    Name: Mapped[str]
    AlbumId: Mapped[int]
    MediaTypeId: Mapped[int]
    GenreId: Mapped[int]
    Composer: Mapped[str]
    Milliseconds: Mapped[int]
    Bytes: Mapped[int]
    UnitPrice: Mapped[float]

# Option 2: refecting tables: generate Table objects automatically from database

Base.metadata.reflect(engine)   # get metadata from database
```

```
class Albums(Base):  # each table is a subclass from the Base table
    __table__ = Base.metadata.tables['albums']

class Artists(Base):
    __table__ = Base.metadata.tables['artists']

class Genres(Base):
    __table__ = Base.metadata.tables['genres']

class InvoiceItems(Base):
    __table__ = Base.metadata.tables['invoice_items']
```

3.2. Statement 1

```
select trackid, name, milliseconds/1000
from tracks
order by milliseconds desc
limit 5;
```

In [ ]:
```
stmt = select(Tracks.TrackId,Tracks.Name, (Tracks.Milliseconds/1000).label('Seconds')) \
        .order_by(Tracks.Milliseconds.desc()) \
        .limit(5)

tracks = session.execute(stmt)

print(f'{"ID":<5} {"Name":<30} {"Seconds":>7}')
print('-'*44)

for track in tracks:
    print(type(track))
    print(f'{track.TrackId:<5} {track.Name:<30} {round(track.Seconds):>7}')
```

```
ID    Name                           Seconds
--------------------------------------------
<class 'sqlalchemy.engine.row.Row'>
2820  Occupation / Precipice            5287
<class 'sqlalchemy.engine.row.Row'>
3224  Through a Looking Glass           5089
<class 'sqlalchemy.engine.row.Row'>
3244  Greetings from Earth, Pt. 1       2960
<class 'sqlalchemy.engine.row.Row'>
3242  The Man With Nine Lives           2957
<class 'sqlalchemy.engine.row.Row'>
3227  Battlestar Galactica, Pt. 2       2956
```

3.3. Statement 2

```sql
select ar.Name,sum(ii.Quantity*ii.UnitPrice)
from tracks t join albums al
on t.AlbumId = al.AlbumId
join artists ar on al.ArtistId = ar.ArtistId
join invoice_items ii  on t.TrackId = ii.TrackId
group by ar.Name
order by sum(ii.Quantity*ii.UnitPrice) desc
limit 10;
```

In [ ]:

```python
from sqlalchemy import func

stmt = select(Artists.Name,func.sum(InvoiceItems.Quantity * InvoiceItems.UnitPrice).label('Sales')) \
        .select_from(Tracks) \
        .join(Albums, Tracks.AlbumId == Albums.AlbumId) \
        .join(InvoiceItems, Tracks.TrackId == InvoiceItems.TrackId) \
        .join(Artists, Albums.ArtistId == Artists.ArtistId) \
        .group_by(Artists.Name) \
        .order_by(func.sum(InvoiceItems.Quantity * InvoiceItems.UnitPrice).desc()) \
        .limit(10)

rows = session.execute(stmt)

print(f'{"Name":<30} {"Sales":>7}')
print('-'*38)
for row in rows:
    print(f'{row.Name:<30} {round(row.Sales,2):7.2f}' )
```

```
Name                            Sales
--------------------------------------
Iron Maiden                    138.60
U2                             105.93
Metallica                       90.09
Led Zeppelin                    86.13
Lost                            81.59
The Office                      49.75
Os Paralamas Do Sucesso         44.55
Deep Purple                     43.56
Faith No More                   41.58
Eric Clapton                    39.60
```

3.4. Statement 3

```sql
update tracks
set unitprice = unitprice * (1 + ?/100.0)
where GenreId in (select genreid from genres where name = ?);
```

In [ ]:
```python
# Option 1: with update statement
from sqlalchemy import update, bindparam
stmt = update(Tracks).values(UnitPrice=Tracks.UnitPrice * (1 + bindparam("pct"))).where(Tracks.GenreId.in_(select(Genres.GenreId).where(Gen
print(stmt) # stringify the statement
session.execute(stmt,{"pct":0.1, "genre":"Pop"})
```

```
UPDATE tracks SET "UnitPrice"=(tracks."UnitPrice" * (:param_1 + :pct)) WHERE tracks."GenreId" IN (SELECT genres."GenreId"
FROM genres
WHERE genres."Name" = :genre)
```

Out[ ]:
```
<sqlalchemy.engine.cursor.CursorResult at 0x2c594a623c0>
```

In [ ]:
```python
# Option 2: with ORM objects
pct = 0.1
genre = 'Pop'
stmt = select(Tracks).where(Tracks.GenreId.in_(select(Genres.GenreId).where(Genres.Name == genre)))
print(stmt)  # stringify the statement
tracks = session.scalars(stmt)  # with scalars() we receive ORM entities directly

for track in tracks:
    track.UnitPrice = track.UnitPrice * (1 + pct)
```

```
SELECT tracks."TrackId", tracks."Name", tracks."AlbumId", tracks."MediaTypeId", tracks."GenreId", tracks."Composer", tracks."Milliseconds",
tracks."Bytes", tracks."UnitPrice"
FROM tracks
WHERE tracks."GenreId" IN (SELECT genres."GenreId"
FROM genres
WHERE genres."Name" = :Name_1)
```

In [ ]:
```python
# Check results
stmt = select(Tracks).where(Tracks.GenreId.in_(select(Genres.GenreId).where(Genres.Name == 'Pop')))
print(stmt)  # stringify the statement
tracks = session.scalars(stmt)  # with scalars() we receive ORM entities directly

print(f'{"ID":<5} {"Name":<40} {"Unitprice":>7}')
print('-'*54)
for track in tracks:
    print(f'{track.TrackId:<5} {track.Name:<40} {track.UnitPrice:7.2f}')
```

```
SELECT tracks."TrackId", tracks."Name", tracks."AlbumId", tracks."MediaTypeId", tracks."GenreId", tracks."Composer", tracks."Milliseconds",
tracks."Bytes", tracks."UnitPrice"
FROM tracks
WHERE tracks."GenreId" IN (SELECT genres."GenreId"
FROM genres
WHERE genres."Name" = :Name_1)
```

ID	Name	Unitprice
323	Dig-Dig, Lambe-Lambe (Ao Vivo)	1.20
324	Pererê	1.20
325	TriboTchan	1.20
326	Tapa Aqui, Descobre Ali	1.20
327	Daniela	1.20
328	Bate Lata	1.20
329	Garotas do Brasil	1.20
330	Levada do Amor (Ailoviu)	1.20
331	Lavadeira	1.20
332	Reboladeira	1.20
333	É que Nessa Encarnação Eu Nasci Manga	1.20
334	Reggae Tchan	1.20
335	My Love	1.20
336	Latinha de Cerveja	1.20
3253	Instant Karma	1.20
3254	#9 Dream	1.20
3255	Mother	1.20
3256	Give Peace a Chance	1.20
3257	Cold Turkey	1.20
3258	Whatever Gets You Thru the Night	1.20
3259	I'm Losing You	1.20
3260	Gimme Some Truth	1.20
3261	Oh, My Love	1.20
3262	Imagine	1.20
3263	Nobody Told Me	1.20
3264	Jealous Guy	1.20
3265	Working Class Hero	1.20
3266	Power to the People	1.20
3267	Imagine	1.20
3268	Beautiful Boy	1.20
3269	Isolation	1.20
3270	Watching the Wheels	1.20
3271	Grow Old With Me	1.20
3272	Gimme Some Truth	1.20
3273	[Just Like] Starting Over	1.20
3274	God	1.20
3275	Real Love	1.20
3467	Intro / Stronger Than Me	1.20
3468	You Sent Me Flying / Cherry	1.20
3469	F**k Me Pumps	1.20

```
3470  I Heard Love Is Blind                        1.20
3471  (There Is) No Greater Love (Teo Licks)       1.20
3472  In My Bed                                    1.20
3473  Take the Box                                 1.20
3474  October Song                                 1.20
3475  What Is It About Men                         1.20
3476  Help Yourself                                1.20
3477  Amy Amy Amy (Outro)                          1.20
```

In [ ]:
```
session.rollback()
```

In [ ]:
```
session.commit()
```

3.5 Statement 4

```
INSERT INTO artists('Name') VALUES('Camille')
```

- Bij gebruik van het ORM is het Session-object verantwoordelijk voor het construeren van Insert-constructies en het uitzenden ervan in een transactie.
- De manier waarop we de Session instrueren dit te doen is door object entries toe te voegen.
- De Session zorgt er dan voor dat deze nieuwe entries naar de database worden verzonden wanneer ze nodig zijn, met behulp van een proces dat bekend staat als een flush.

In [ ]:
```
camille = Artists(Name='Camille')
session.add(camille)
```

In [ ]:
```
# Check results
stmt = select(Artists).order_by(Artists.ArtistId.desc()).limit(1)
print(stmt)  # stringify the statement
last_artist = session.execute(stmt).scalar_one() # convert result of select statement to object

print(f'{"ArtistId":<10} {"Name":<30}')
print('-'*32)
print(f'{last_artist.ArtistId:<10} {last_artist.Name:<30}' )
```

```
SELECT artists."ArtistId", artists."Name"
FROM artists ORDER BY artists."ArtistId" DESC
 LIMIT :param_1
ArtistId   Name
--------------------------------
284        Camille
```

In [ ]:
```
session.rollback()
```

In [ ]:
```
session.commit()
```

3.6 Statement 5

```
DELETE FROM artists WHERE name='Camille'
```

In [ ]:
```
stmt = select(Artists).where(Artists.Name == 'Camille')
result = session.scalars(stmt).all()  # with scalars() we receive ORM entities directly.
if len(result) > 0:
    print('Artist Camille exists')
    camille = result[0]   # get first artist with name Camille
    session.delete(camille)
    print('Artist Camille deleted')
else:
    print('Artist Camille does not exist')
```

```
Artist Camille does not exist
```

In [ ]:
```
# Check results
stmt = select(Artists).order_by(Artists.ArtistId.desc()).limit(1)

last_artist = session.execute(stmt).scalar_one() # convert result of select statement to object

print(f'{"ArtistId":<10} {"Name":<30}')
print('-'*32)
print(f'{last_artist.ArtistId:<10} {last_artist.Name:<30}' )
```

```
ArtistId   Name
--------------------------------
275        Philip Glass Ensemble
```

In [ ]:
```
session.rollback()
```

In [ ]:
```
session.commit()
```

3.7. Afsluiten

In [ ]:
```
conn.close()
```

3.8. Voor- en nadelen van ORM

- `+` SQL syntax check door Python --> fouten ontdekt bij het ontwikkelen
- `+` Porteerbaar tussen databanksystemen
- `+` Integratie met Python-objecten.
- `-` Extra syntax moet aangeleerd worden.

- - Risico op trage code (vooral bij Optie 2: rechtstreeks gebruik van objecten) omdat men niet meer stilstaat bij gegenereerde SQL-commando's:
  - SELECT *: onnodig veel kolommen ophalen
  - teveel "round-trips" naar de database, door bijv. in een lus updates uit te voeren i.p.v. in één update-statement.

4. Conclusies

- Gebruik "Plain SQL"
  - om snel resultaat te hebben
  - als je vertrouwd bent met de SQL-taal
  - voor erg complexe query's die moeilijk om te zetten zijn naar SQLAlchemy
- Gebruik SQL Alchemy (ORM of Core)
  - voor productiewaardige software
- Gebruik SQL Alchemy ORM
  - als object-oriëntatie een must is
  - als onderdeel van een grotere applicatie
  - als "state" of geheugen tussen calls belangrijk is
  - maar trap niet in de "performantie-val": zorg ervoor dat je goed weet wat er gebeurt op de database