1. DATA QUALITY CHECKS (SQL CODE) - YAN

1.1 Correct data inconsistencies

1.1.1 Check duplicate rows

```
SELECT agentid, COUNT(*) FROM agents
GROUP BY agentid
HAVING COUNT(*) > 1

SELECT entertainerid, memberid,COUNT(*) FROM entertainer_members em
GROUP BY entertainerid, memberid
HAVING COUNT(*) > 1
```

After running queries like these to every table, **no duplicate records** were found in any of the tables. All primary keys are unique. This dataset maintains strong referential integrity and does not require deduplication.

1.1.2 Check inconsistent data formats

```
SELECT * FROM agents
WHERE agtstate !~ '^[A-Z]{2}$'

SELECT * FROM agents
WHERE agtzipcode !~ '^[0-9]{5}$'

SELECT * FROM agents
WHERE agtphonenumber !~ '^[0-9]{3}-[0-9]{4}$'

SELECT entertainerid, entwebpage FROM entertainers
WHERE entwebpage IS NOT NULL
 AND entwebpage !~ '^www\.[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'

 SELECT entertainerid, entemailaddress FROM entertainers
WHERE entemailaddress IS NOT NULL
 AND entemailaddress !~ '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'
```

After checking the formats of states, zipcode, phonenumber, email addresses and webpage URLs, there are **no invalid** patterns detected. Although street addresses show minor differences, it is normal to see that in daily life. Overall, the dataset shows no formatting errors that would impact analysis.

1.1.3 Check invalid entries

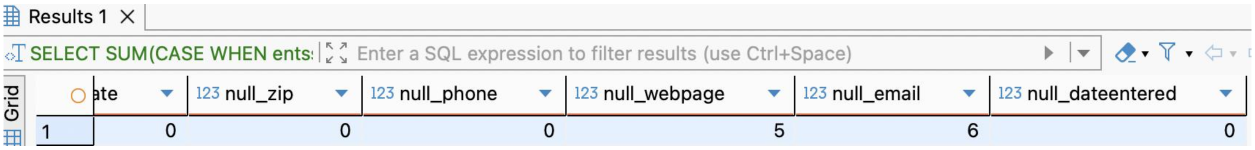
```
SELECT * FROM agents
WHERE salary < 0 OR commissionrate < 0

SELECT * FROM engagements
WHERE contractprice < 0


SELECT * FROM Engagements
WHERE EndDate < StartDate


SELECT * FROM entertainers
WHERE dateentered > CURRENT_DATE
```

All queries returned zero records, indicating that the dataset contains **no invalid entries**. The numerical fields, date fields, and business fields are all valid.

1.2 Deal with Missing Values, Surprising Values or Outliers

1.2.1 Check NULLs in critical columns

```
SELECT
  SUM(CASE WHEN entertainerid     IS NULL THEN 1 ELSE 0 END) AS null_entertainerid,
  SUM(CASE WHEN entstagename      IS NULL THEN 1 ELSE 0 END) AS null_stagename,
  SUM(CASE WHEN entssn            IS NULL THEN 1 ELSE 0 END) AS null_ssn,
  SUM(CASE WHEN entstreetaddress  IS NULL THEN 1 ELSE 0 END) AS null_address,
  SUM(CASE WHEN entcity           IS NULL THEN 1 ELSE 0 END) AS null_city,
  SUM(CASE WHEN entstate          IS NULL THEN 1 ELSE 0 END) AS null_state,
  SUM(CASE WHEN entzipcode        IS NULL THEN 1 ELSE 0 END) AS null_zip,
  SUM(CASE WHEN entphonenumber    IS NULL THEN 1 ELSE 0 END) AS null_phone,
  SUM(CASE WHEN entwebpage        IS NULL THEN 1 ELSE 0 END) AS null_webpage,
  SUM(CASE WHEN entemailaddress   IS NULL THEN 1 ELSE 0 END) AS null_email,
  SUM(CASE WHEN dateentered       IS NULL THEN 1 ELSE 0 END) AS null_dateentered
FROM entertainers
```



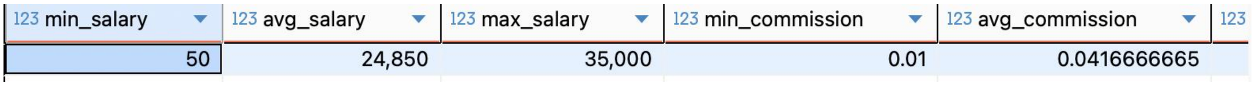
SELECT SUM(CASE WHEN ents:

	ate	null_zip	null_phone	null_webpage	null_email	null_dateentered
1	0	0	0	5	6	0

After checking all the tables for null values, there are only 2 columns containing NULLs in the entertainers table: **5 NULLs in entwebpage, 6 NULLs in entemailaddress.** These columns represent optional contact information, indicating there is some missing contact information. The dataset is not sparse.

1.2.2 Check surprising values

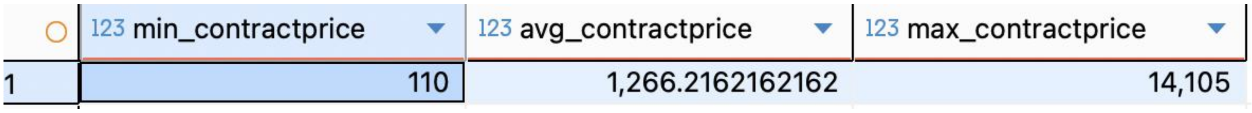
```
SELECT
  MIN(salary) AS min_salary,
  AVG(salary) AS avg_salary,
  MAX(salary) AS max_salary,
  MIN(commissionrate) AS min_commission,
  AVG(commissionrate) AS avg_commission,
  MAX(commissionrate) AS max_commission
FROM agents
```



min_salary	avg_salary	max_salary	min_commission	avg_commission	123
50	24,850	35,000	0.01	0.0416666665	

The minimum salary of agents is only 50, and the commission rate is lower than average. This shows a huge difference, this row could be an outlier and needs to be deleted.

```
SELECT
  MIN(contractprice) AS min_contractprice,
  AVG(contractprice) AS avg_contractprice,
  MAX(contractprice) AS max_contractprice
FROM engagements
```



○	min_contractprice	avg_contractprice	max_contractprice
1	110	1,266.2162162162	14,105

This range of contract prices is relatively large, and because the values vary continuously rather than clustering around specific levels, these differences likely reflect the natural variation in engagement types rather than true outliers. So this table may have no outliers.

```
SELECT
  MIN(startdate) AS min_startdate,
  MAX(startdate) AS max_startdate,
  MIN(enddate) AS min_enddate,
  MAX(enddate) AS max_enddate
FROM engagements
```

The dates contain no incorrect or illogical values.

1.3 Conclusion - Data quality notes

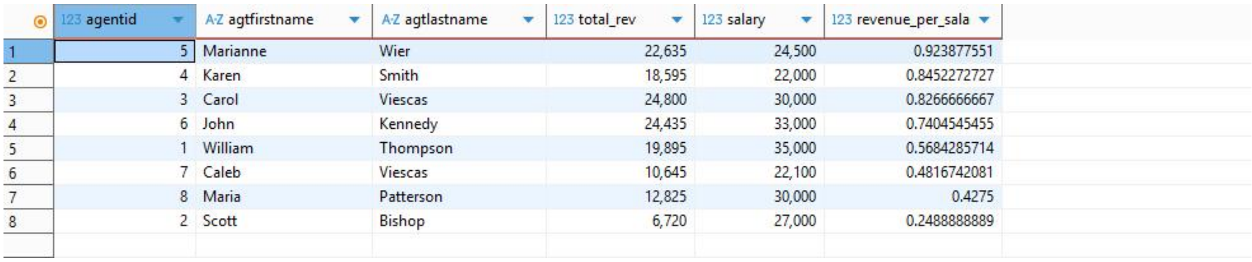
Overall, the dataset is clean and well-structured. All tables contain no duplicates, invalid entries, or formatting inconsistencies. The only missing values identified are in the *email* and *website* columns of the entertainers table.

In examining the numeral outliers, all values fall within reasonable ranges except for one agent whose *salary* and *commission rate* are unusually low compared to the average.

2. UNIVARIATE ANALYSIS (SQL) - MIKE & JOSHUA

2.1 Agent’s profit

```
with Agent_profit as (select a.agentid, a.agtfirstname, a.agtlastname, sum(e.contractprice) as
total_rev
from agents a
join engagements e
on a.agentid =e.agentid
group by a.agentid, a.agtfirstname, a.agtlastname
order by total_rev desc)
select a.agentid, a.agtfirstname, a.agtlastname, total_rev, salary, total_rev/salary as
Revenue_per_Salary
from agents a
join Agent_profit ap
on a.agentid = ap.agentid
order by total_rev/salary desc
```



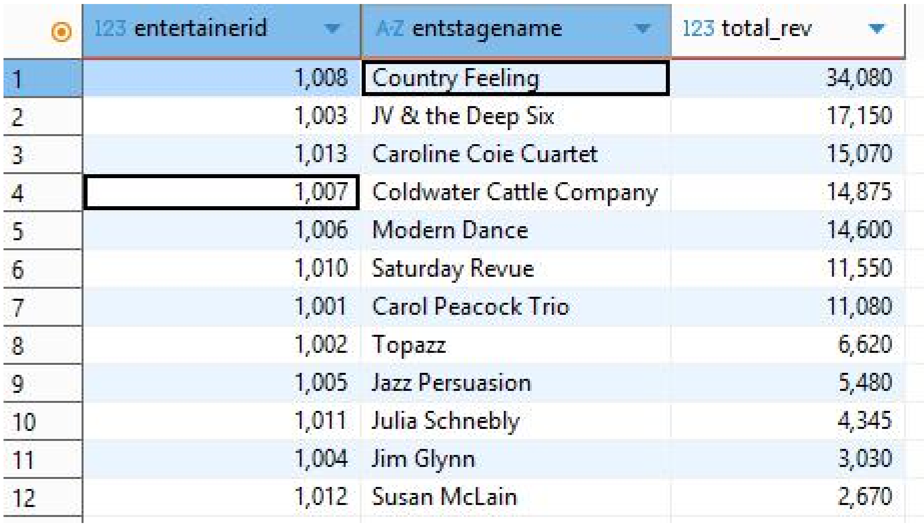
	agentid	agtfirstname	agtlastname	total_rev	salary	revenue_per_sala
1	5	Marianne	Wier	22,635	24,500	0.923877551
2	4	Karen	Smith	18,595	22,000	0.8452272727
3	3	Carol	Viescas	24,800	30,000	0.8266666667
4	6	John	Kennedy	24,435	33,000	0.7404545455
5	1	William	Thompson	19,895	35,000	0.5684285714
6	7	Caleb	Viescas	10,645	22,100	0.4816742081
7	8	Maria	Patterson	12,825	30,000	0.4275
8	2	Scott	Bishop	6,720	27,000	0.2488888889

I first aimed to identify which agent generated the highest total revenue, so I calculated each agent’s total contract revenue and ranked them accordingly. After reviewing the results, I realized that revenue alone doesn’t reflect how efficiently an agent performs, since salaries vary. Therefore, I incorporated salary data and created a Revenue_per_Salary metric (total revenue divided by salary) to evaluate how much revenue each agent generates for every dollar the company pays them. This gave me a clearer view of both top earners and the most cost-efficient agents.

2.2 Solo Artists vs. Bands

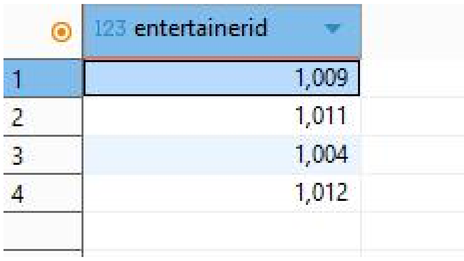
```
--entertainers' profit
```

```
select e.entertainerid, e.entstagename, sum(en.contractprice) as total_rev
from engagements en
join entertainers e
on e.entertainerid = en.entertainerid
group by e.entertainerid , e.entstagename
order by total_rev desc
```



	entertainerid	entstagename	total_rev
1	1,008	Country Feeling	34,080
2	1,003	JV & the Deep Six	17,150
3	1,013	Caroline Coie Cuartet	15,070
4	1,007	Coldwater Cattle Company	14,875
5	1,006	Modern Dance	14,600
6	1,010	Saturday Revue	11,550
7	1,001	Carol Peacock Trio	11,080
8	1,002	Topazz	6,620
9	1,005	Jazz Persuasion	5,480
10	1,011	Julia Schnebly	4,345
11	1,004	Jim Glynn	3,030
12	1,012	Susan McLain	2,670

```
--Solo Artists's id
select e.entertainerid from entertainers  e
join entertainer_members em
on e.entertainerid = em.entertainerid
join members m
on em.memberid = m.memberid
group by e.entertainerid
having count(*) = 1
```

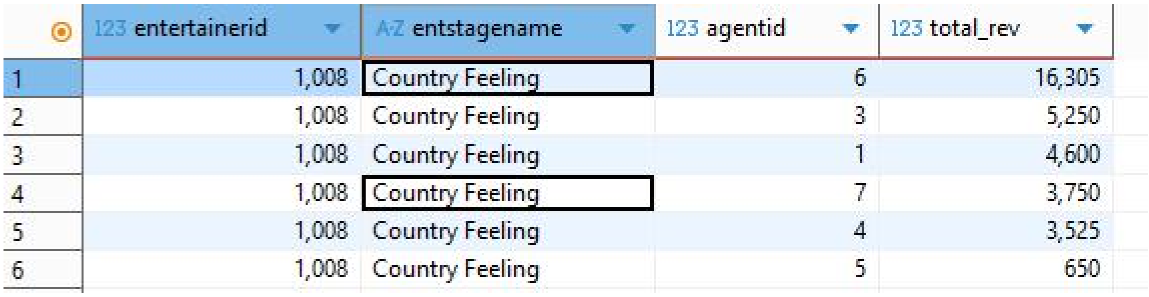


	entertainerid
1	1,009
2	1,011
3	1,004
4	1,012

To compare performance across different types of entertainers, I first calculated the total revenue generated by each entertainer using their contract earnings. This allowed me to rank all entertainers by revenue, regardless of whether they were solo artists or bands. Next, I used a separate query to identify solo entertainers by selecting those with exactly one member in the entertainer_members table. By comparing the solo list with the full revenue rankings, I could contrast the performance of solo artists versus bands. After examining the results, a clear pattern emerged: bands consistently generated higher total revenue than solo artists.

2.3 Identifying the Top-Contributing Agent for a High-Revenue Band

```
select e.entertainerid , e.entstagename, a.agentid, sum(en.contractprice) as total_rev
from engagements en
join entertainers e
on e.entertainerid = en.entertainerid
join agents a
on en.agentid = a.agentid
where e.entertainerid = '1008'
group by e.entertainerid , e.entstagename, a.agentid
order by entertainerid, total_rev desc
```

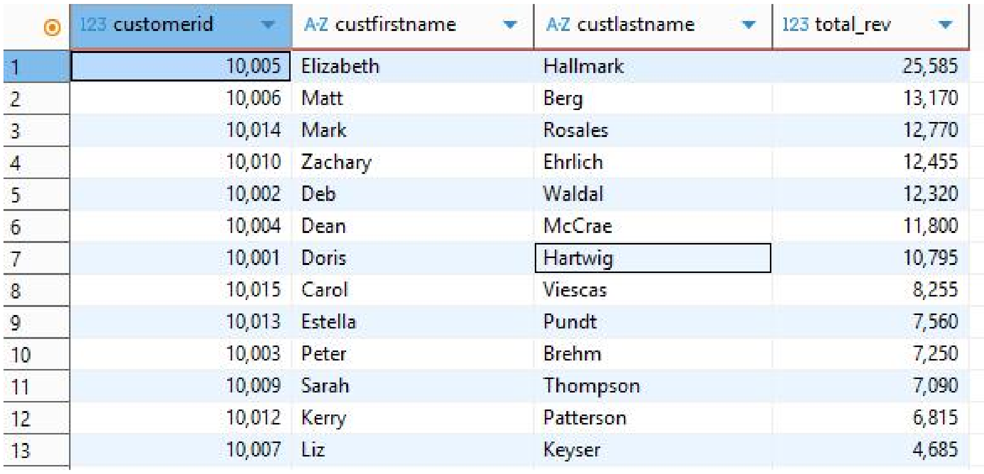


	entertainerid	entstagename	agentid	total_rev
1	1,008	Country Feeling	6	16,305
2	1,008	Country Feeling	3	5,250
3	1,008	Country Feeling	1	4,600
4	1,008	Country Feeling	7	3,750
5	1,008	Country Feeling	4	3,525
6	1,008	Country Feeling	5	650

To analyze agent–entertainer matching efficiency at the individual band level, I focused on the highest-earning band and examined how much revenue each agent generated for that band. By ranking agents according to the total revenue they brought in, I could identify which agent was most effective in generating profitable engagements specifically for this high-performing band.

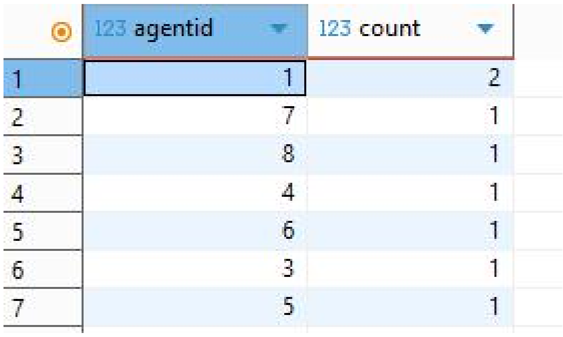
2.4 Understanding Our Most Valuable Customer's Preferences

```
select c.customerid , c.custfirstname, c.custlastname, sum(en.contractprice) as total_rev
from engagements en
join customers c
on c.customerid = en.customerid
group by c.customerid , c.custfirstname, c.custlastname
order by total_rev desc
```



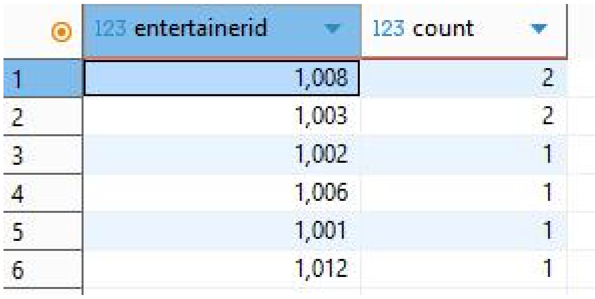
	customerid	custfirstname	custlastname	total_rev
1	10,005	Elizabeth	Hallmark	25,585
2	10,006	Matt	Berg	13,170
3	10,014	Mark	Rosales	12,770
4	10,010	Zachary	Ehrlich	12,455
5	10,002	Deb	Waldal	12,320
6	10,004	Dean	McCrae	11,800
7	10,001	Doris	Hartwig	10,795
8	10,015	Carol	Viescas	8,255
9	10,013	Estella	Pundt	7,560
10	10,003	Peter	Brehm	7,250
11	10,009	Sarah	Thompson	7,090
12	10,012	Kerry	Patterson	6,815
13	10,007	Liz	Keyser	4,685

```
--Agent's close with our most valuable customers
select agentid, count(agentid) from engagements e
where customerid = '10005'
group by agentid
order by count(*) desc
```



	agentid	count
1	1	2
2	7	1
3	8	1
4	4	1
5	6	1
6	3	1
7	5	1

```
--What our most valuable customers like
select e.entertainerid, count(e.entertainerid) from engagements e
where customerid = '10005'
group by e.entertainerid
order by count(*) desc
```

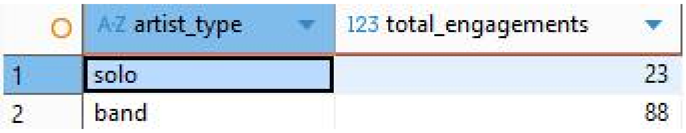


	entertainerid	count
1	1,008	2
2	1,003	2
3	1,002	1
4	1,006	1
5	1,001	1
6	1,012	1

To analyze customer–agent–entertainer relationships, I first identified the company’s highest-spending customer by summing contract revenue across all engagements. After determining the most valuable customer, I examined whether this customer consistently worked with the same agent by counting the number of engagements handled by each agent for that customer. The results showed no dominant agent, indicating that the customer does not appear to have a strong preference for any particular representative.Next, I evaluated whether the customer showed consistent preferences for specific entertainers. By grouping engagements by entertainer ID and counting how often the customer hired each performer, I again found no clear pattern. This suggests that even our top customer spreads their bookings across different entertainers rather than repeatedly selecting the same act. Overall, the analysis indicates that our highest-value customer does not demonstrate strong loyalty to any specific agent or entertainer, implying their purchasing decisions may be more event-driven or situational rather than preference-based.

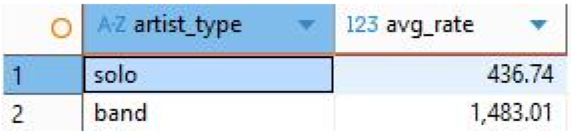
2.5 Understanding Why Bands Have Higher Revenue and Engagement

```
--Band vs. Solo Engagements
with solo_artists as (
 select e.entertainerid
 from entertainers e
 join entertainer_members em on e.entertainerid = em.entertainerid
 group by e.entertainerid
 having count(*) = 1)
select
 case when e.entertainerid in (select entertainerid from solo_artists) then 'solo' else 'band' end as artist_type,
 count(*) as total_engagements
from engagements e
join entertainers ent on e.entertainerid = ent.entertainerid
group by artist_type
```



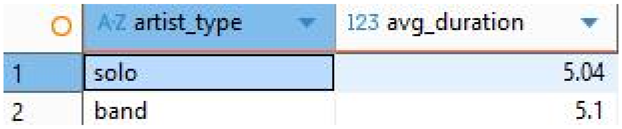
	artist_type	total_engagements
1	solo	23
2	band	88

```
--Band vs. Solo Rates
with solo_artists as (
 select e.entertainerid
 from entertainers e
 join entertainer_members em on e.entertainerid = em.entertainerid
 group by e.entertainerid
 having count(*) = 1)
select
 case
	when e.entertainerid in (select entertainerid from solo_artists) then 'solo'
	else 'band'
	end as artist_type,
 round(avg(e.contractprice),2) as avg_rate
from engagements e
join entertainers ent on e.entertainerid = ent.entertainerid
group by artist_type
```



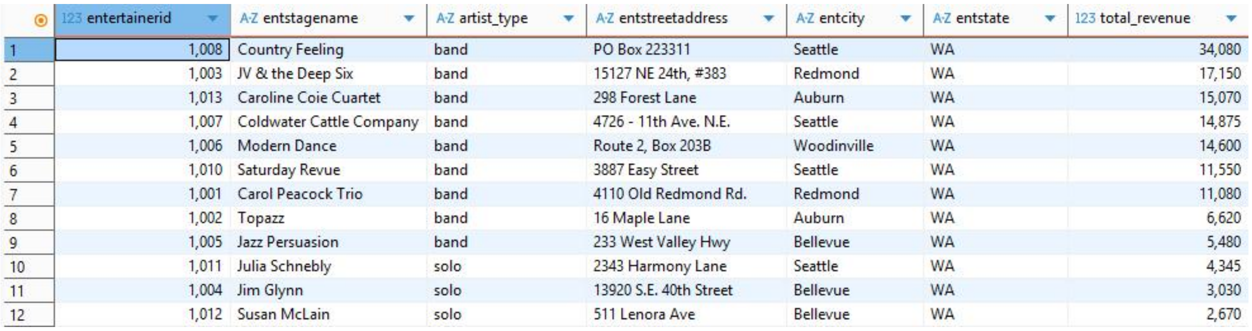
	artist_type	avg_rate
1	solo	436.74
2	band	1,483.01

```
--Duration Analysis
with solo_artists as (
 select e.entertainerid
 from entertainers e
 join entertainer_members em on e.entertainerid = em.entertainerid
 group by e.entertainerid
 having count(*) = 1)
select
 case when e.entertainerid in (select entertainerid from solo_artists) then 'solo' else 'band' end as artist_type,
 round(avg(e.enddate - e.startdate),2) as avg_duration
from engagements e
join entertainers ent on e.entertainerid = ent.entertainerid
group by artist_type
```



	artist_type	avg_duration
1	solo	5.04
2	band	5.1

```
--Location Analysis
with solo_artists as (
 select entertainerid
 from entertainer_members
 group by entertainerid
 having count(memberid) = 1)
select e.entertainerid, e.entstagename,
 case
   when e.entertainerid in (select entertainerid from solo_artists) then 'solo'
   else 'band'
 end as artist_type,
 e.entstreetaddress, e.entcity, e.entstate, sum(en.contractprice) as total_revenue
from engagements en
join entertainers e
on en.entertainerid = e.entertainerid
group by e.entertainerid, e.entstagename, artist_type, e.entstreetaddress, e.entcity, e.entstate
order by total_revenue desc
```



	entertainerid	entstagename	artist_type	entstreetaddress	entcity	entstate	total_revenue
1	1,008	Country Feeling	band	PO Box 223311	Seattle	WA	34,080
2	1,003	JV & the Deep Six	band	15127 NE 24th, #383	Redmond	WA	17,150
3	1,013	Caroline Coie Cuartet	band	298 Forest Lane	Auburn	WA	15,070
4	1,007	Coldwater Cattle Company	band	4726 - 11th Ave. N.E.	Seattle	WA	14,875
5	1,006	Modern Dance	band	Route 2, Box 203B	Woodinville	WA	14,600
6	1,010	Saturday Revue	band	3887 Easy Street	Seattle	WA	11,550
7	1,001	Carol Peacock Trio	band	4110 Old Redmond Rd.	Redmond	WA	11,080
8	1,002	Topazz	band	16 Maple Lane	Auburn	WA	6,620
9	1,005	Jazz Persuasion	band	233 West Valley Hwy	Bellevue	WA	5,480
10	1,011	Julia Schnebly	solo	2343 Harmony Lane	Seattle	WA	4,345
11	1,004	Jim Glynn	solo	13920 S.E. 40th Street	Bellevue	WA	3,030
12	1,012	Susan McLain	solo	511 Lenora Ave	Bellevue	WA	2,670

<u>NOTES ON BAND VS. SOLO PERFORMERS</u>

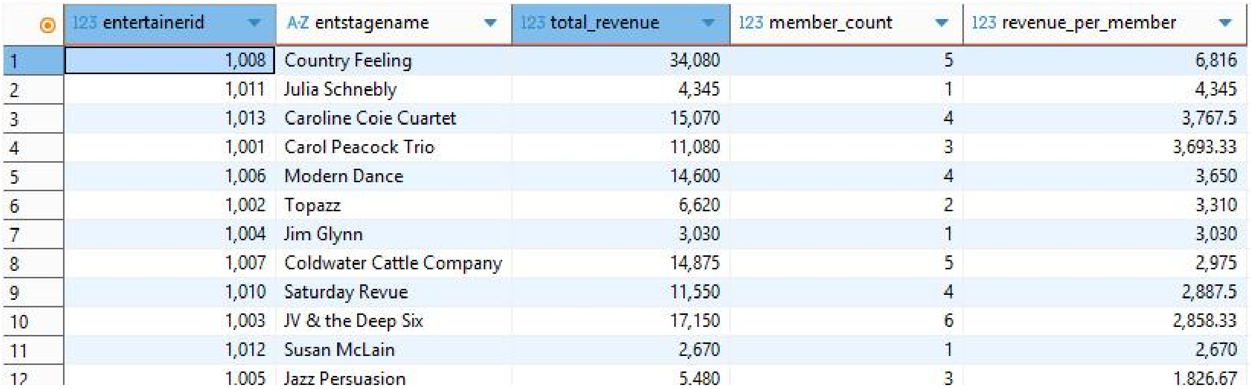
With all of these things in mind, it can be seen that location and duration have little to no impact on the drastic revenue differences between solo and band-based artists (despite a slight difference in duration and a slight indication that locations like Bellevue offer lower total revenue possibility). Instead, a big rationale as to why bands are bringing in more revenue than solo artists appears to be the lesser number of total engagements and lower average rate value.

To combat this, the organization could consider marketing pushes for their solo artists to increase both awareness and usage of these artists. They should also consider the possibility of rate and contract restructuring to overcome the lower rates of these performers.

One thing to be cautious of in this area is the price elasticity of consumers. It could be that consumers are not willing to pay as much for solo performers, hence the lowered rates. To combat this risk, they could consider rolling out any price and rate changes in incremental stages.

Another avenue is that the company should focus a majority of their resources to bands as solo artists do not efficiently generate revenue in the same way.

```
--Revenue Per Performer Analysis
with members_per_entertainer as (
 Select entertainerid, count(memberid) as member_count
 from entertainer_members
 group by entertainerid),
revenue_per_entertainer as (
 select e.entertainerid, e.entstagename, sum(en.contractprice) as total_revenue
 from entertainers e
 join engagements en on e.entertainerid = en.entertainerid
 group by e.entertainerid, e.entstagename)
select r.entertainerid,  r.entstagename, r.total_revenue, m.member_count,
 round(r.total_revenue::numeric / m.member_count, 2) as revenue_per_member
from revenue_per_entertainer r
join members_per_entertainer m
on r.entertainerid = m.entertainerid
order by revenue_per_member desc
```



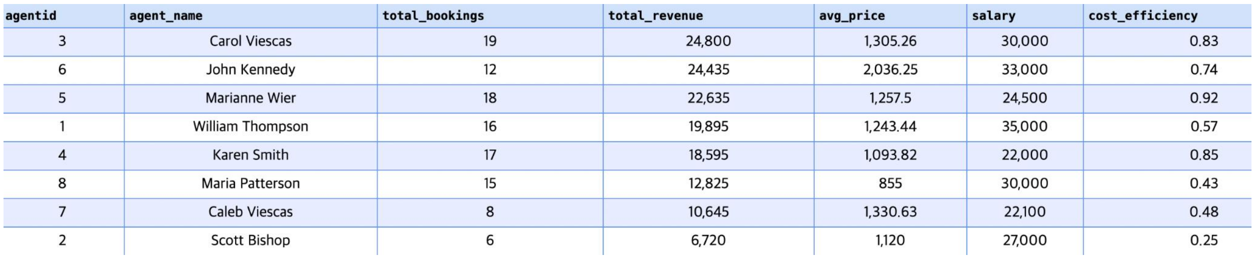
	entertainerid	entstagename	total_revenue	member_count	revenue_per_member
1	1,008	Country Feeling	34,080	5	6,816
2	1,011	Julia Schnebly	4,345	1	4,345
3	1,013	Caroline Coie Cuartet	15,070	4	3,767.5
4	1,001	Carol Peacock Trio	11,080	3	3,693.33
5	1,006	Modern Dance	14,600	4	3,650
6	1,002	Topazz	6,620	2	3,310
7	1,004	Jim Glynn	3,030	1	3,030
8	1,007	Coldwater Cattle Company	14,875	5	2,975
9	1,010	Saturday Revue	11,550	4	2,887.5
10	1,003	JV & the Deep Six	17,150	6	2,858.33
11	1,012	Susan McLain	2,670	1	2,670
12	1,005	Jazz Persuasion	5,480	3	1,826.67

We can even see that the average revenue per member is not impacted by the number of people in the group. This can indicate that a focus should be on groups as each member can bring in revenue for the group as a whole.

3. MULTIVARIATE ANALYSIS (SQL JOINS)- KRYSTAL

3.1 Internal workforce (Agents) performance and efficiency analysis

```
SELECT
 a.agentid,
 concat(a.agtfirstname, ' ', a.agtlastname) as agent_name,
 count(e.engagementnumber) as total_bookings,
 sum(e.contractprice) as total_revenue,
 round(avg(e.contractprice), 2) as avg_price,
 a.salary,
 round(sum(e.contractprice) / nullif(a.salary, 0), 2) as cost_efficiency
FROM agents a
LEFT JOIN engagements e
 on a.agentid = e.agentid
GROUP BY
 a.agentid,
 concat(a.agtfirstname, ' ', a.agtlastname),
 a.salary
having sum(e.contractprice) is not null
ORDER BY total_revenue
desc
```



agentid	agent_name	total_bookings	total_revenue	avg_price	salary	cost_efficiency
3	Carol Viescas	19	24,800	1,305.26	30,000	0.83
6	John Kennedy	12	24,435	2,036.25	33,000	0.74
5	Marianne Wier	18	22,635	1,257.5	24,500	0.92
1	William Thompson	16	19,895	1,243.44	35,000	0.57
4	Karen Smith	17	18,595	1,093.82	22,000	0.85
8	Maria Patterson	15	12,825	855	30,000	0.43
7	Caleb Viescas	8	10,645	1,330.63	22,100	0.48
2	Scott Bishop	6	6,720	1,120	27,000	0.25

The purpose of this query is to evaluate each agent's overall performance and efficiency by analyzing key metrics such as the number of bookings handled, total revenue generated, average contract value, and cost efficiency, calculated by comparing revenue to the agent's salary. By joining the Agents and Engagements tables, we can identify which agents consistently secure high-value contracts and which agents generate strong cost efficiency relative to their compensation.

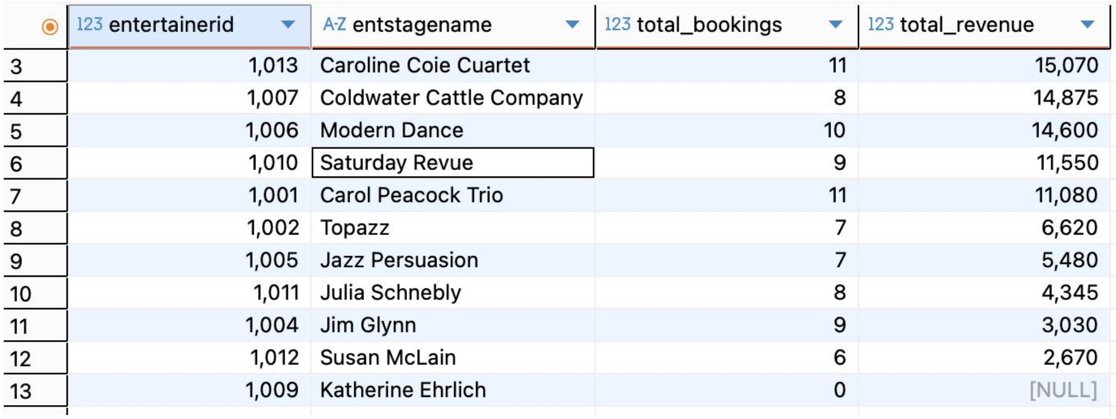
However, in our dataset, salaries are recorded as full annual costs, while engagement revenue reflects only a partial period. Because of this mismatch, traditional ROI does not provide a realistic comparison of performance.

This analysis therefore focuses on cost efficiency rather than classical ROI, allowing the company to distinguish top performers who drive both revenue and efficiency from those who may require further support or performance review.

3.2 Entertainers performance analysis

```
SELECT
    ent.EntertainerID,
    ent.EntStageName,
    COUNT(e.EngagementNumber) AS total_bookings,
    SUM(e.ContractPrice) AS total_revenue
FROM Entertainers ent
```

```
LEFT JOIN Engagements e USING (EntertainerID)
GROUP BY ent.EntertainerID, ent.EntStageName
ORDER BY total_revenue DESC NULLS LAST;
```

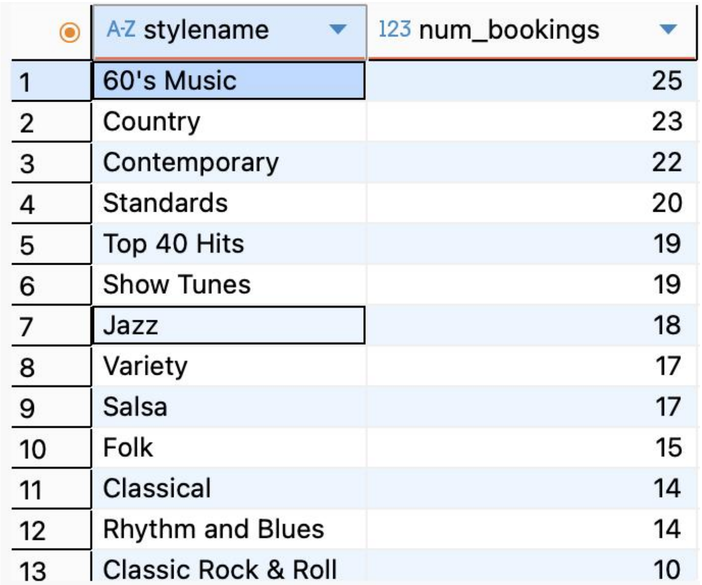


	123 entertainerid	A-Z entstagename	123 total_bookings	123 total_revenue
3	1,013	Caroline Coie Cuartet	11	15,070
4	1,007	Coldwater Cattle Company	8	14,875
5	1,006	Modern Dance	10	14,600
6	1,010	Saturday Revue	9	11,550
7	1,001	Carol Peacock Trio	11	11,080
8	1,002	Topazz	7	6,620
9	1,005	Jazz Persuasion	7	5,480
10	1,011	Julia Schnebly	8	4,345
11	1,004	Jim Glynn	9	3,030
12	1,012	Susan McLain	6	2,670
13	1,009	Katherine Ehrlich	0	[NULL]

This query was designed to evaluate the performance of each entertainer by analyzing their total number of bookings and the total revenue generated across all performances. By joining the Entertainer table and the Performance table, we aimed to identify which entertainers attract the highest demand and which generate the greatest economic value. This helps inform decisions regarding pricing, promotional strategies, and entertainer portfolio management.

3.3 Market and trend analysis based on genre demand

```
SELECT
    ms.stylename,
    COUNT(*) AS num_bookings
FROM engagements e
JOIN entertainers ent
    ON e.entertainerid = ent.entertainerid
JOIN entertainer_styles es
    ON ent.entertainerid = es.entertainerid
JOIN musical_styles ms
    ON es.styleid = ms.styleid
GROUP BY ms.stylename
ORDER BY num_bookings DESC;
```

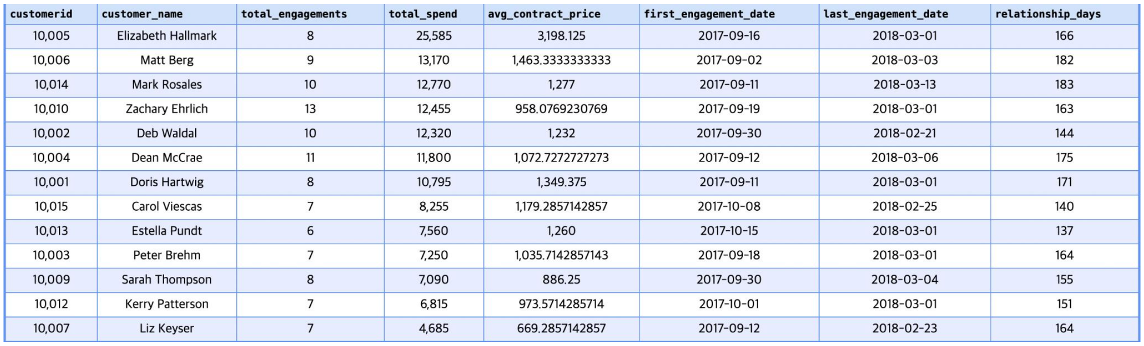


	stylename	num_bookings
1	60's Music	25
2	Country	23
3	Contemporary	22
4	Standards	20
5	Top 40 Hits	19
6	Show Tunes	19
7	Jazz	18
8	Variety	17
9	Salsa	17
10	Folk	15
11	Classical	14
12	Rhythm and Blues	14
13	Classic Rock & Roll	10

This query aimed to identify the music genres most frequently requested by customers. By joining the Engagements, Entertainers, entertainer_styles, and musical_styles tables, we sought to capture broad market demand across various genres. Understanding reservation patterns by genre is considered helpful for grasping customer preferences and overall trends within the entertainment market. This analysis aimed to guide strategic decisions related to discovering new talent, expanding portfolios, and marketing efforts by identifying the styles currently driving engagement.

3.4 Customer perspective analysis focusing on LTV, relationship duration, and high-value clients

```
SELECT
    c.customerid,
    CONCAT(c.custfirstname, ' ', c.custlastname) AS customer_name,
    COUNT(e.engagementnumber) AS total_engagements,
    SUM(e.contractprice) AS total_spend,
    AVG(e.contractprice) AS avg_contract_price,
    MIN(e.startdate) AS first_engagement_date, --Relationship span
    MAX(e.enddate) AS last_engagement_date,
   MAX(e.enddate) - MIN(e.startdate) AS relationship_days
FROM customers c
LEFT JOIN engagements e
   ON c.customerid = e.customerid
GROUP BY
   c.customerid, CONCAT(c.custfirstname, ' ', c.custlastname)
HAVING SUM(e.contractprice) IS NOT NULL
ORDER BY total_spend DESC;
```



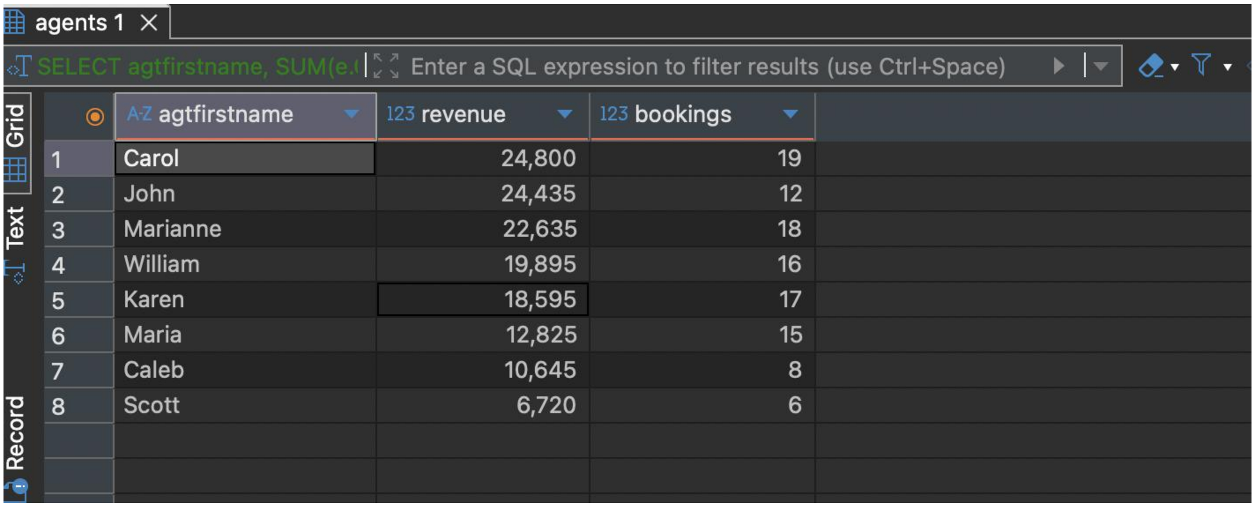
customerid	customer_name	total_engagements	total_spend	avg_contract_price	first_engagement_date	last_engagement_date	relationship_days
10,005	Elizabeth Hallmark	8	25,585	3,198.125	2017-09-16	2018-03-01	166
10,006	Matt Berg	9	13,170	1,463.3333333333	2017-09-02	2018-03-03	182
10,014	Mark Rosales	10	12,770	1,277	2017-09-11	2018-03-13	183
10,010	Zachary Ehrlich	13	12,455	958.0769230769	2017-09-19	2018-03-01	163
10,002	Deb Waldal	10	12,320	1,232	2017-09-30	2018-02-21	144
10,004	Dean McCrae	11	11,800	1,072.7272727273	2017-09-12	2018-03-06	175
10,001	Doris Hartwig	8	10,795	1,349.375	2017-09-11	2018-03-01	171
10,015	Carol Viescas	7	8,255	1,179.2857142857	2017-10-08	2018-02-25	140
10,013	Estella Pundt	6	7,560	1,260	2017-10-15	2018-03-01	137
10,003	Peter Brehm	7	7,250	1,035.7142857143	2017-09-18	2018-03-01	164
10,009	Sarah Thompson	8	7,090	886.25	2017-09-30	2018-03-04	155
10,012	Kerry Patterson	7	6,815	973.5714285714	2017-10-01	2018-03-01	151
10,007	Liz Keyser	7	4,685	669.2857142857	2017-09-12	2018-02-23	164

The purpose of this query was to identify high-value customers by analyzing their total spending, engagement frequency, and the duration of their relationship with the company. By combining the Customers table and the Engagements table, we aimed to derive key metrics that help quantify Customer Lifetime Value (CLV), such as total spending, average contract price, and the time interval between a customer's first engagement and their most recent engagement. This analysis aimed to distinguish one-time users from repeat customers and identify those contributing most significantly to long-term business stability. The ultimate goal was to support more targeted customer relationship management and retention strategies.

4. KEY BUSINESS QUESTIONS (with SQL)-ROHIT

Q1. Who are the top agents by revenue?

```
SELECT
    a.AgentName,
    SUM(e.ContractPrice) AS revenue,
    COUNT(*) AS bookings
FROM Agents a
JOIN Engagements e USING (AgentID)
GROUP BY a.AgentName
ORDER BY revenue DESC
LIMIT 10;
```

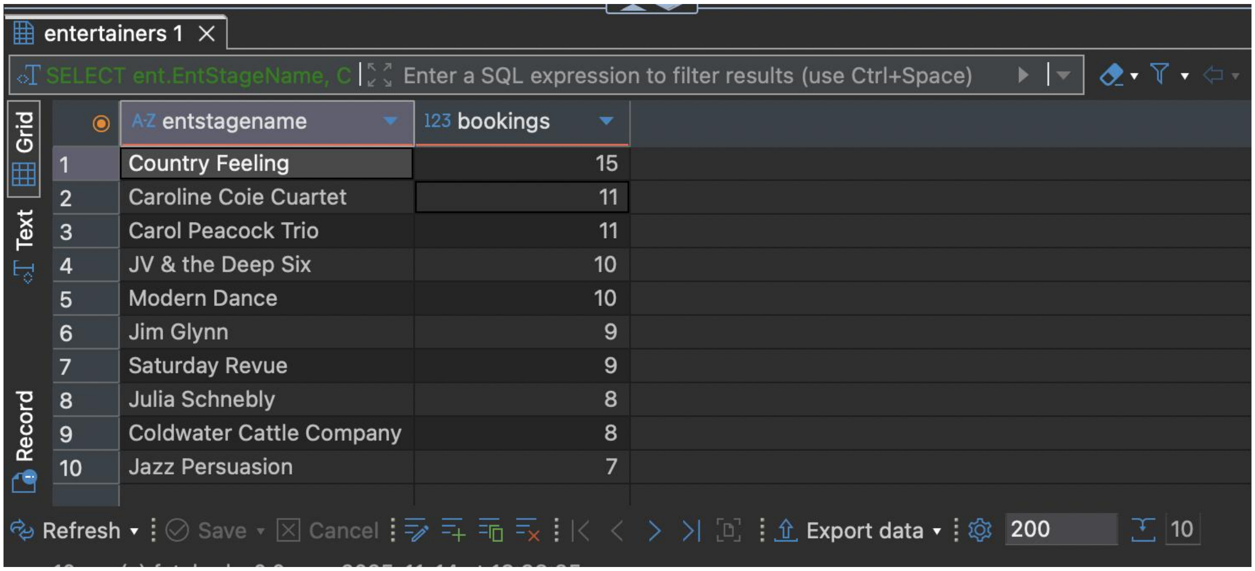

agents 1
SELECT agtfirstname, SUM(e.(
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
Text
Record

	agtfirstname	revenue	bookings
1	Carol	24,800	19
2	John	24,435	12
3	Marianne	22,635	18
4	William	19,895	16
5	Karen	18,595	17
6	Maria	12,825	15
7	Caleb	10,645	8
8	Scott	6,720	6

Q2. Which entertainers are most booked?

```
SELECT
    ent.EntStageName,
    COUNT(e.EngagementNumber) AS bookings
FROM Entertainers ent
JOIN Engagements e USING (EntertainerID)
GROUP BY ent.EntStageName
ORDER BY bookings DESC
```

```
LIMIT 10;
```

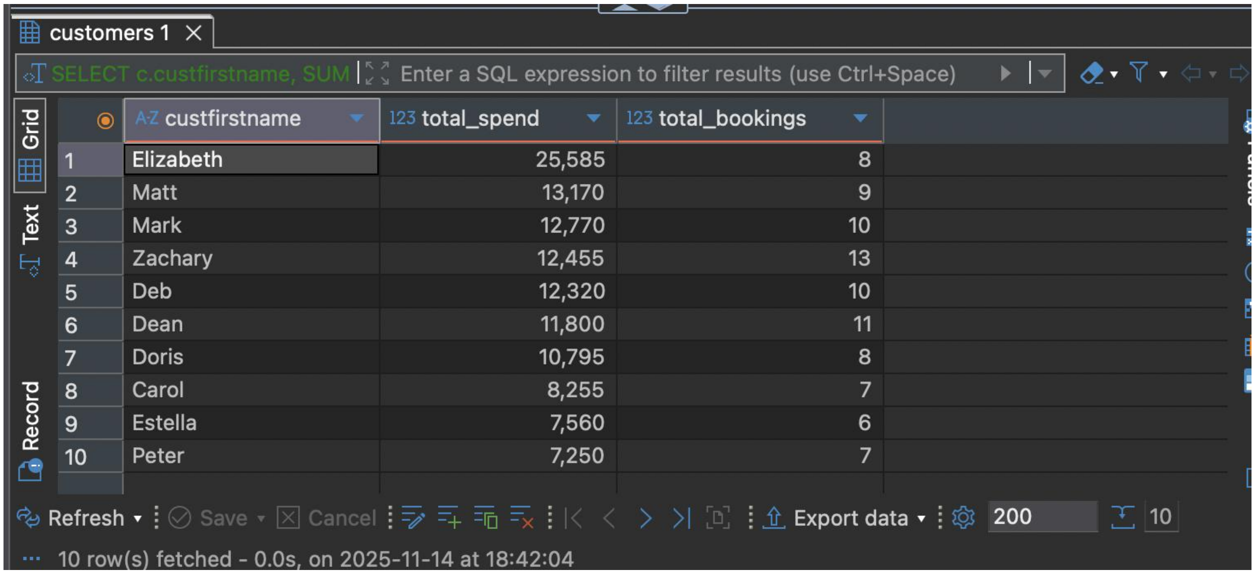


entertainers 1
SELECT ent.EntStageName, C
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
Text
Record
Refresh
Save
Cancel
Export data
200
10

	entstagename	bookings
1	Country Feeling	15
2	Caroline Coie Cuartet	11
3	Carol Peacock Trio	11
4	JV & the Deep Six	10
5	Modern Dance	10
6	Jim Glynn	9
7	Saturday Revue	9
8	Julia Schnebly	8
9	Coldwater Cattle Company	8
10	Jazz Persuasion	7

Q3. Who are the highest-spending customers?

```
SELECT
    c.CustomerName,
    SUM(e.ContractPrice) AS total_spend,
    COUNT(*) AS total_bookings
FROM Customers c
JOIN Engagements e USING (CustomerID)
GROUP BY c.CustomerID
ORDER BY total_spend DESC
LIMIT 10;
```



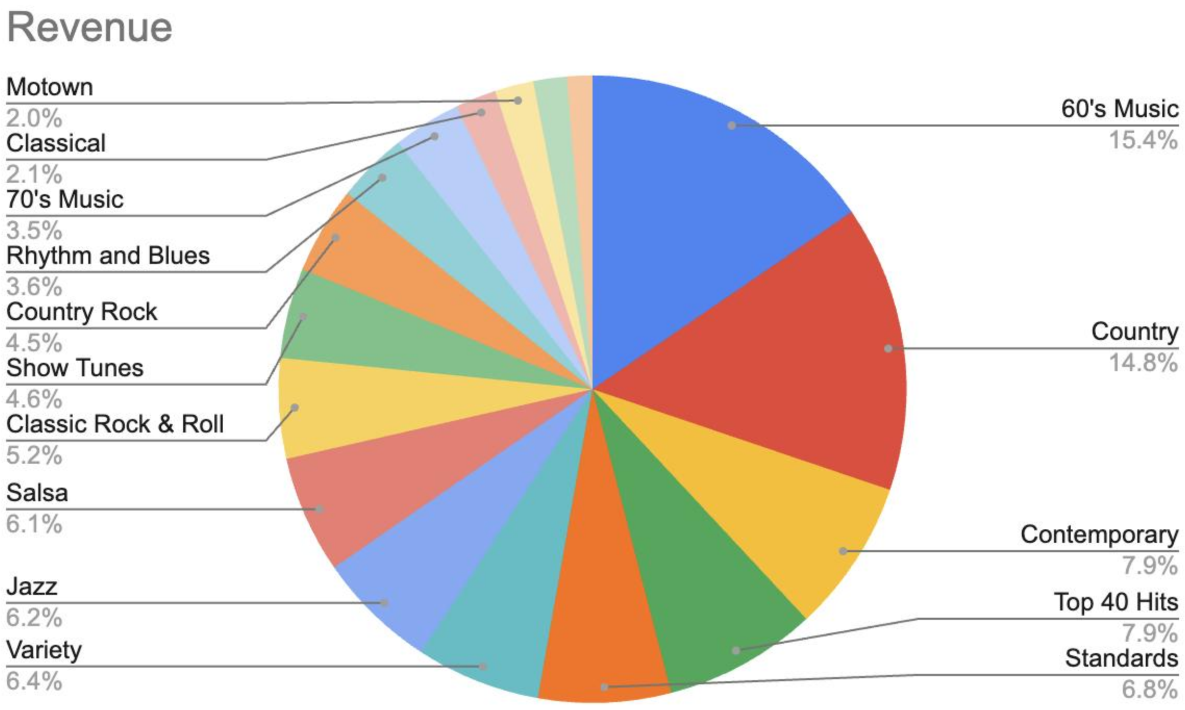
customers 1
SELECT c.custfirstname, SUM
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
Text
Record
Refresh
Save
Cancel
Export data
200
10

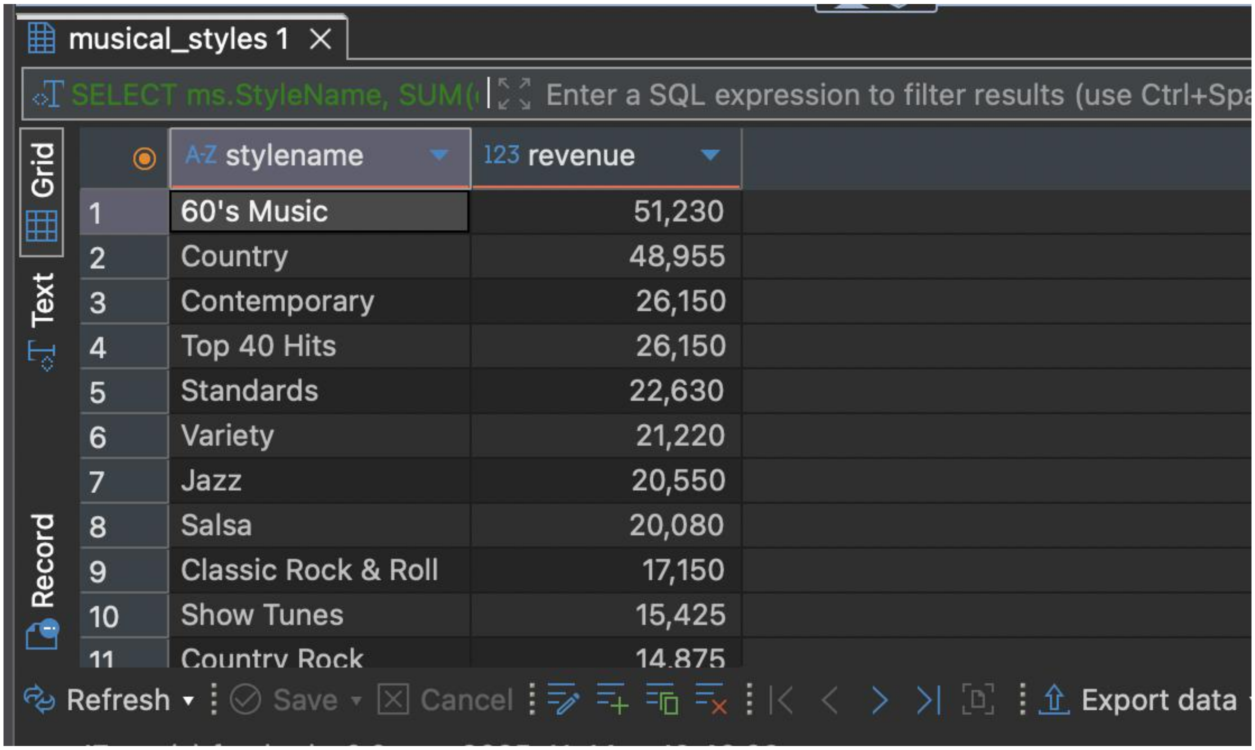
	custfirstname	total_spend	total_bookings
1	Elizabeth	25,585	8
2	Matt	13,170	9
3	Mark	12,770	10
4	Zachary	12,455	13
5	Deb	12,320	10
6	Dean	11,800	11
7	Doris	10,795	8
8	Carol	8,255	7
9	Estella	7,560	6
10	Peter	7,250	7

10 row(s) fetched - 0.0s, on 2025-11-14 at 18:42:04

Q4. Which musical styles generate the most revenue?

```
SELECT
    ms.StyleName,
    SUM(e.ContractPrice) AS revenue
FROM Engagements e
JOIN Entertainer_Styles es USING (EntertainerID)
JOIN Musical_Styles ms USING (StyleID)
GROUP BY ms.StyleName
ORDER BY revenue DESC;
```



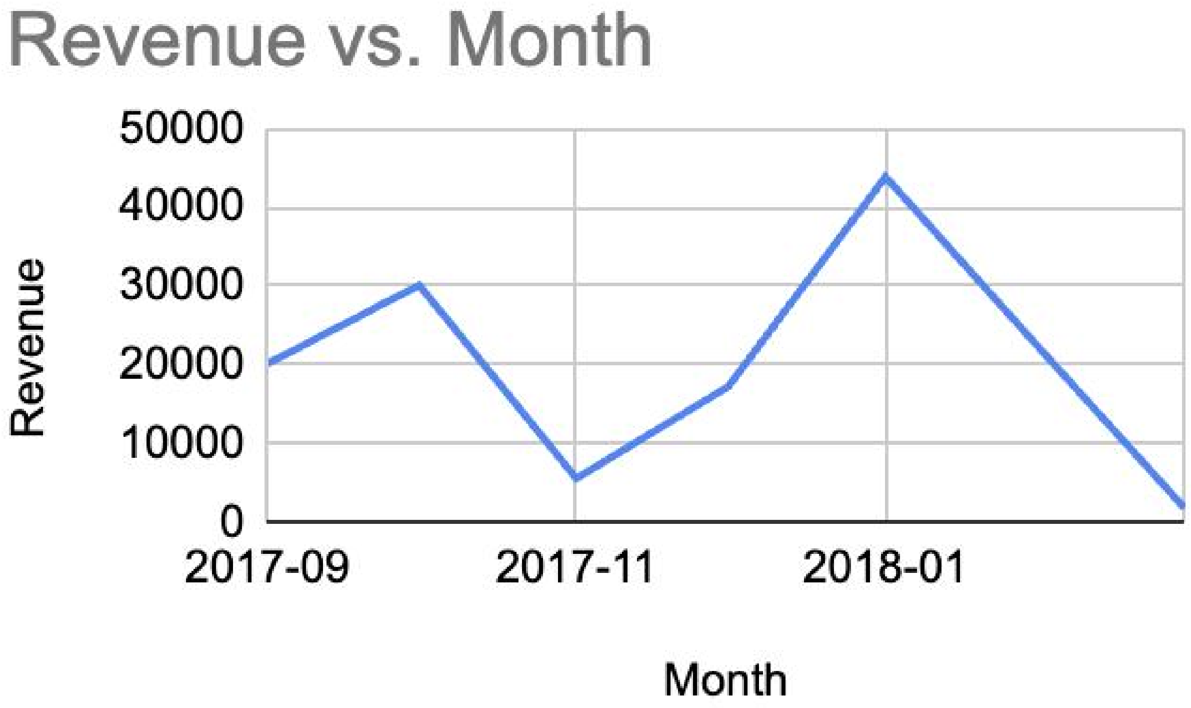


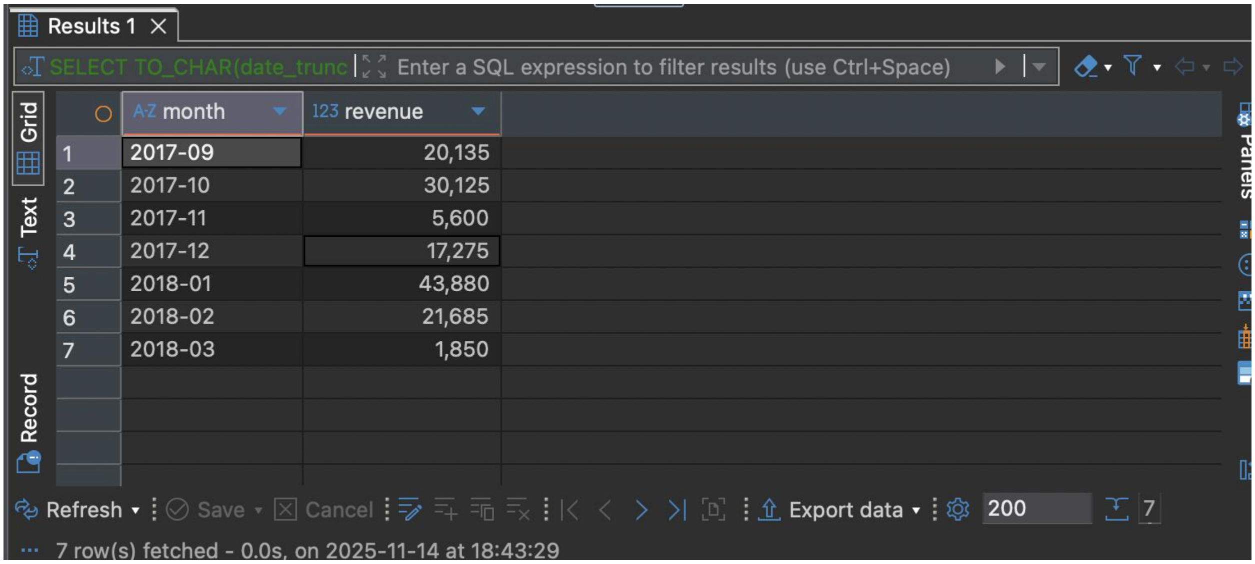
SELECT ms.StyleName, SUM(

	stylename	revenue
1	60's Music	51,230
2	Country	48,955
3	Contemporary	26,150
4	Top 40 Hits	26,150
5	Standards	22,630
6	Variety	21,220
7	Jazz	20,550
8	Salsa	20,080
9	Classic Rock & Roll	17,150
10	Show Tunes	15,425
11	Country Rock	14,875

Q5. Monthly revenue trend

```
SELECT
    TO_CHAR(date_trunc('month', StartDate), 'YYYY-MM') AS month,
    SUM(ContractPrice) AS revenue
FROM Engagements
GROUP BY date_trunc('month', StartDate)
ORDER BY date_trunc('month',
StartDate);
```





SELECT TO_CHAR(date_trunc

Enter a SQL expression to filter results (use Ctrl+Space)

	month	revenue
1	2017-09	20,135
2	2017-10	30,125
3	2017-11	5,600
4	2017-12	17,275
5	2018-01	43,880
6	2018-02	21,685
7	2018-03	1,850

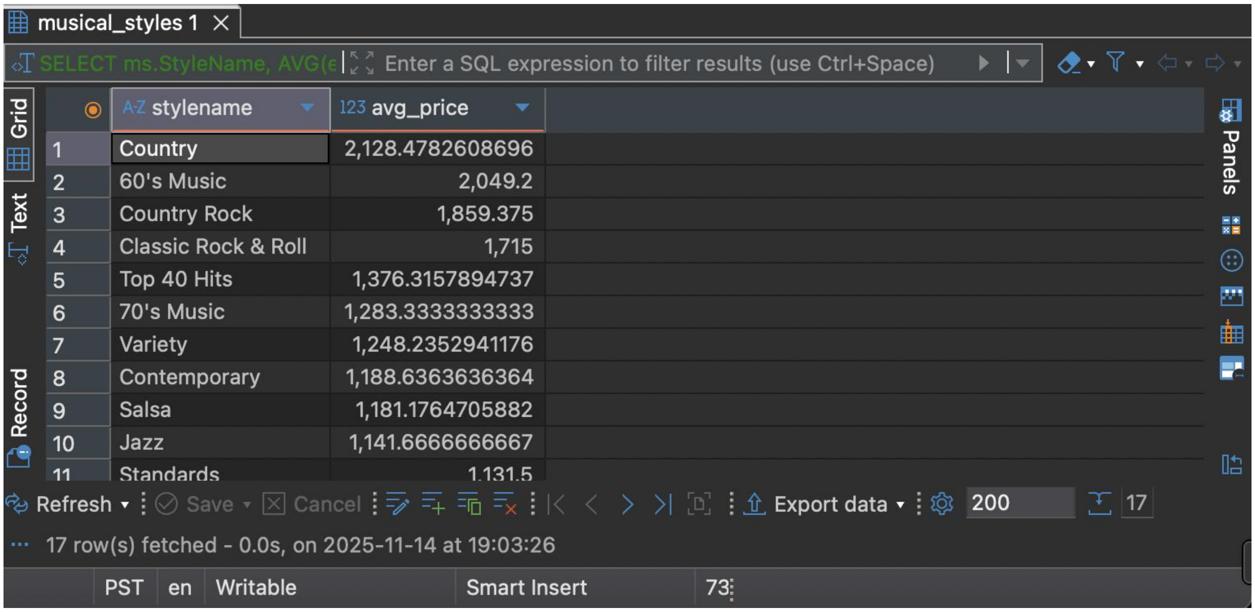
7 row(s) fetched - 0.0s, on 2025-11-14 at 18:43:29

Q6. Which factors influence high contract prices?

You can start with summary relationships:

By style:

```
SELECT ms.StyleName, AVG(e.ContractPrice) AS avg_price
FROM Engagements e
JOIN Entertainer_Styles es USING (EntertainerID)
JOIN Musical_Styles ms USING (StyleID)
GROUP BY ms.StyleName
ORDER BY avg_price DESC;
```

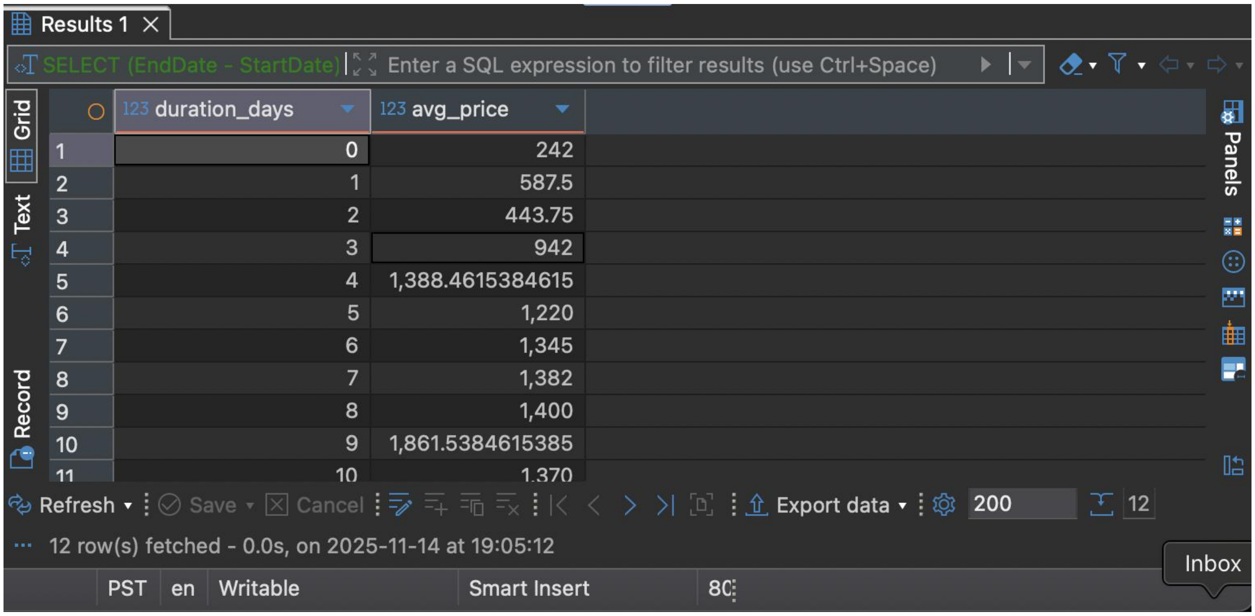


	stylename	avg_price
1	Country	2,128.4782608696
2	60's Music	2,049.2
3	Country Rock	1,859.375
4	Classic Rock & Roll	1,715
5	Top 40 Hits	1,376.3157894737
6	70's Music	1,283.3333333333
7	Variety	1,248.2352941176
8	Contemporary	1,188.6363636364
9	Salsa	1,181.1764705882
10	Jazz	1,141.6666666667
11	Standards	1,131.5

```
17 row(s) fetched - 0.0s, on 2025-11-14 at 19:03:26
```

By duration:

```
SELECT
    (DATEDIFF(EndDate, StartDate)) AS duration_days,
    AVG(ContractPrice) AS avg_price
FROM Engagements
GROUP BY duration_days
ORDER BY duration_days;
```



SELECT (EndDate - StartDate)

Enter a SQL expression to filter results (use Ctrl+Space)

	duration_days	avg_price
1	0	242
2	1	587.5
3	2	443.75
4	3	942
5	4	1,388.4615384615
6	5	1,220
7	6	1,345
8	7	1,382
9	8	1,400
10	9	1,861.5384615385
11	10	1,370

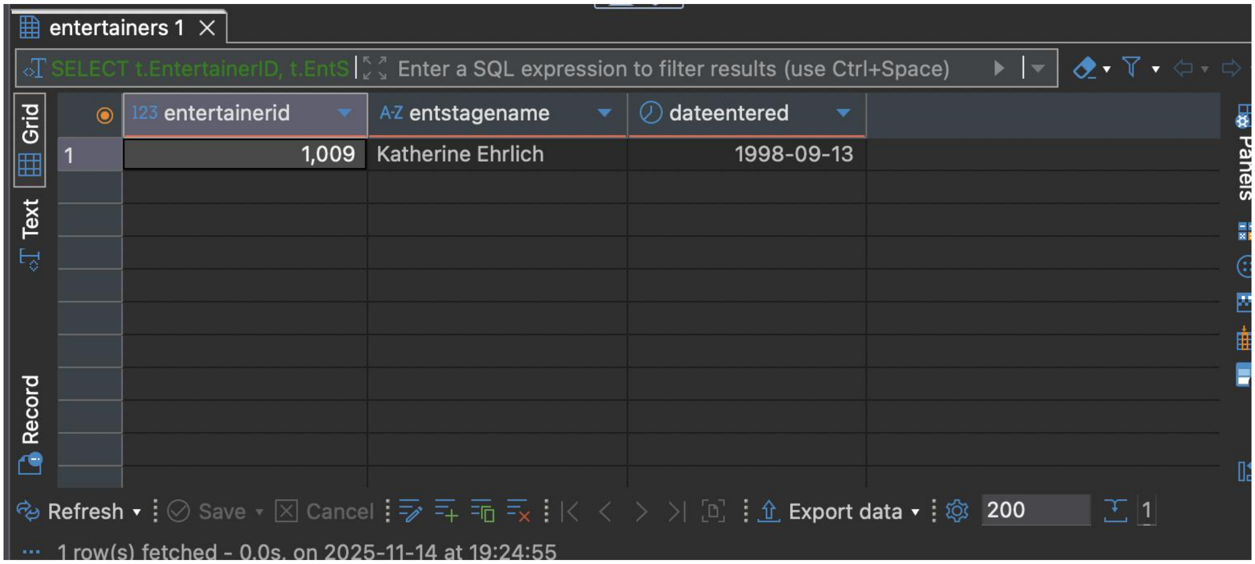
12 row(s) fetched - 0.0s, on 2025-11-14 at 19:05:12

Q7. Which entertainers on our roster are being wasted?

Business Question:
"We spend time and money recruiting entertainers. Are there any who have never been booked?"

SQL (PostgreSQL):

```
SELECT
  t.EntertainerID,
  t.EntStageName,
  t.DateEntered
FROM Entertainers AS t
LEFT JOIN Engagements AS e
  ON t.EntertainerID = e.EntertainerID
WHERE e.EngagementNumber IS NULL
ORDER BY t.DateEntered;
```



entertainers 1
SELECT t.EntertainerID, t.EntS
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
Text
Record
Panels
Refresh
Save
Cancel
Export data
200
1

	entertainerid	entstagename	dateentered
1	1,009	Katherine Ehrlich	1998-09-13

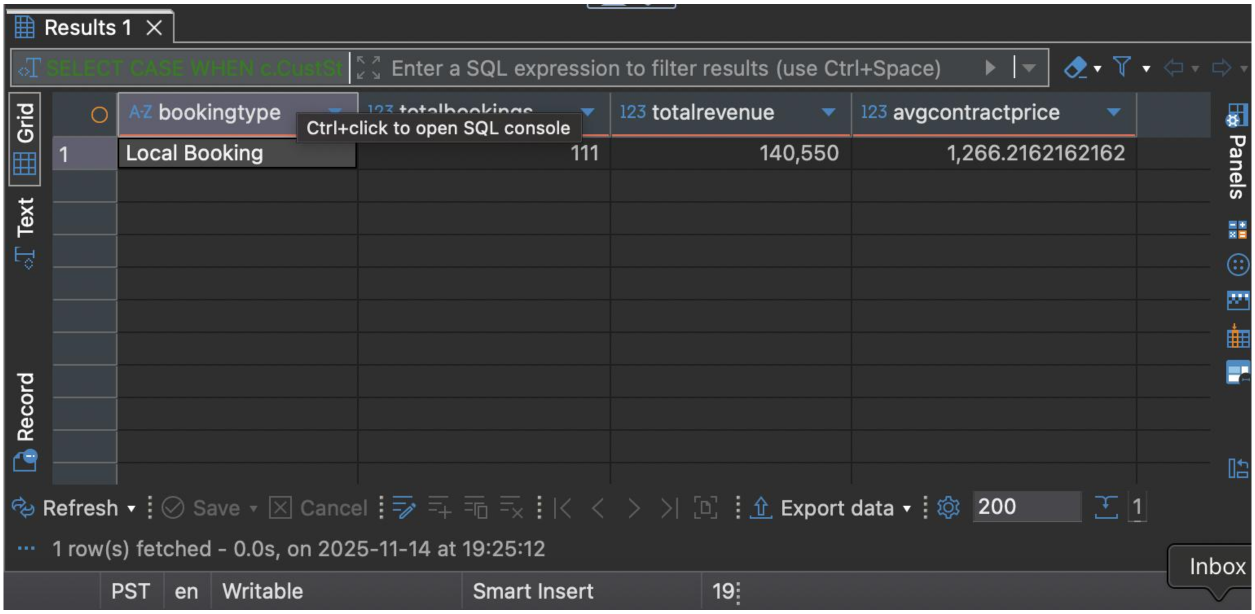
1 row(s) fetched - 0.0s, on 2025-11-14 at 19:24:55

Q8. Do customers book local or non-local talent?

Business Question:
"Do customers in a state book entertainers from the same state, or do they book non-local talent?"

SQL (PostgreSQL):

```
SELECT
  CASE
    WHEN c.CustState = t.EntState THEN 'Local Booking'
    ELSE 'Non-Local Booking'
  END AS BookingType,
  COUNT(e.EngagementNumber) AS TotalBookings,
  SUM(e.ContractPrice) AS TotalRevenue,
  AVG(e.ContractPrice) AS AvgContractPrice
FROM Engagements AS e
JOIN Customers AS c ON e.CustomerID = c.CustomerID
JOIN Entertainers AS t ON e.EntertainerID = t.EntertainerID
GROUP BY BookingType;
```


Results 1
SELECT CASE WHEN c.CustSt
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
Text
Record
Panels
Ctrl+click to open SQL console
Refresh
Save
Cancel
Export data
200
1
PST
en
Writable
Smart Insert
19
Inbox

	bookingtype	totalbookings	totalrevenue	avgcontractprice
1	Local Booking	111	140,550	1,266.2162162162

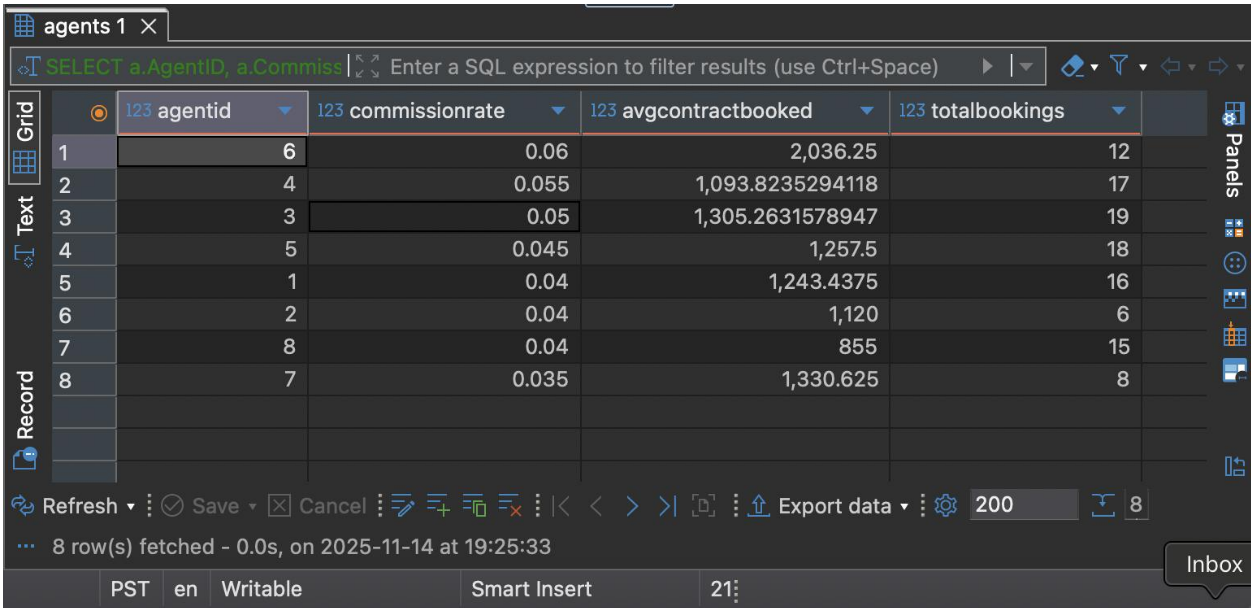
```
1 row(s) fetched - 0.0s, on 2025-11-14 at 19:25:12
```

Q9. Do agent incentives align with business goals?

Business Question:
"Do agents with high commission rates book higher-value gigs?"

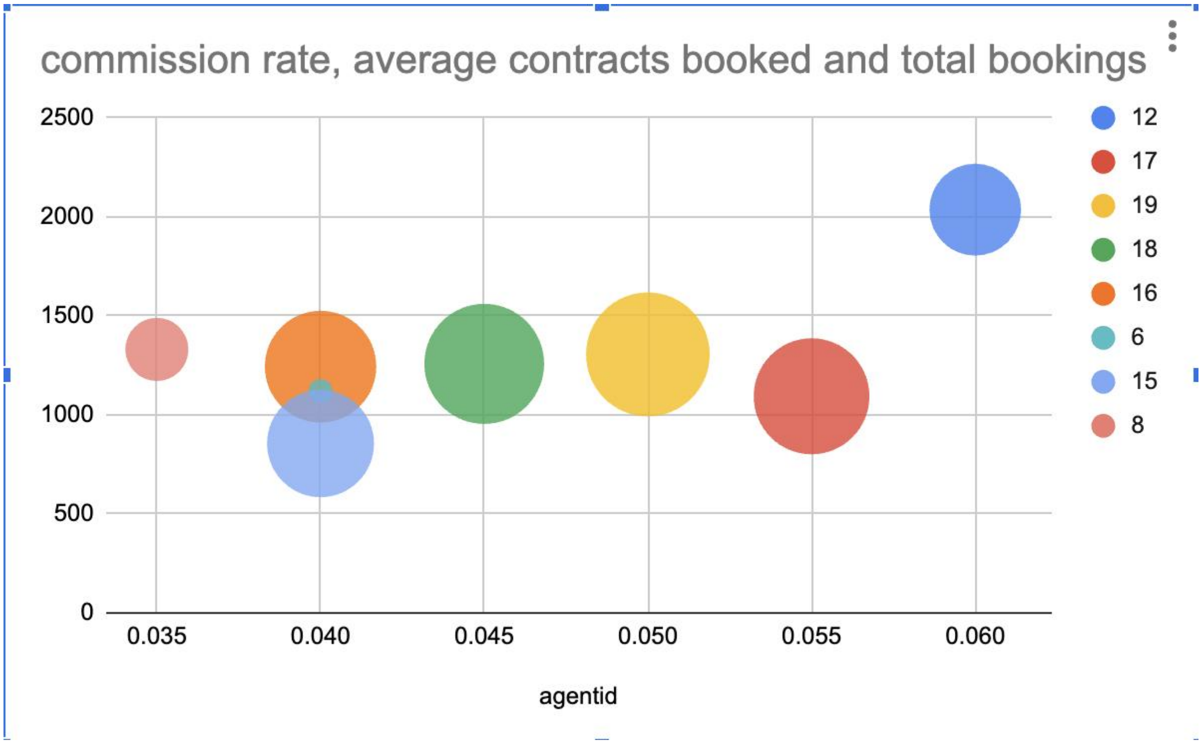
SQL (PostgreSQL):

```
SELECT
  a.AgentID,
  a.CommissionRate,
  AVG(e.ContractPrice) AS AvgContractBooked,
  COUNT(e.EngagementNumber) AS TotalBookings
FROM Agents AS a
JOIN Engagements AS e ON a.AgentID = e.AgentID
GROUP BY
  a.AgentID,
  a.CommissionRate
ORDER BY a.CommissionRate DESC;
```



	agentid	commissionrate	avgcontractbooked	totalbookings
1	6	0.06	2,036.25	12
2	4	0.055	1,093.8235294118	17
3	3	0.05	1,305.2631578947	19
4	5	0.045	1,257.5	18
5	1	0.04	1,243.4375	16
6	2	0.04	1,120	6
7	8	0.04	855	15
8	7	0.035	1,330.625	8

```
8 row(s) fetched - 0.0s, on 2025-11-14 at 19:25:33
```

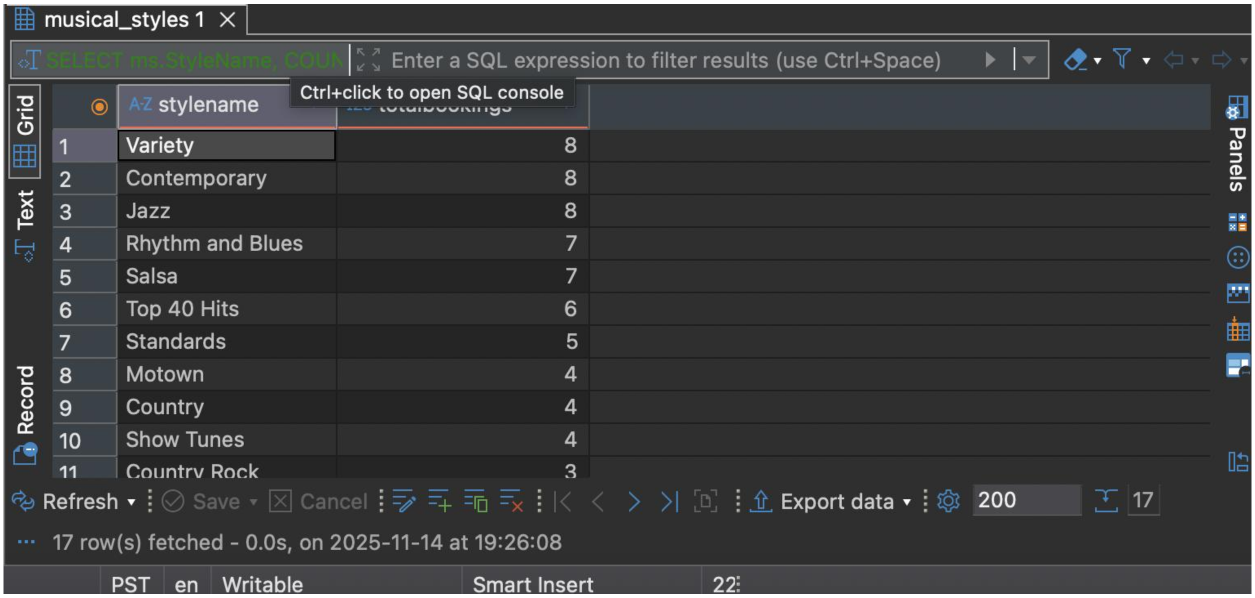


Q10. Do customers book what they say they like?

Business Question:
"For customers who say they like 'Jazz', what styles do they actually book?"

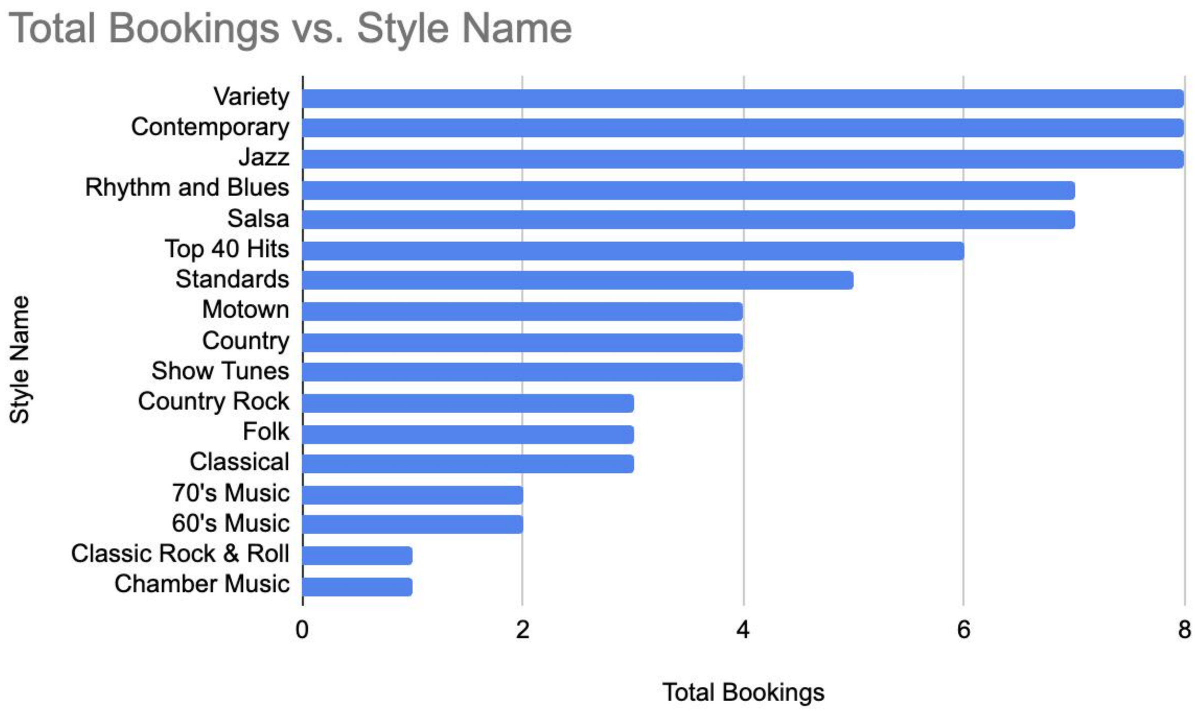
SQL (PostgreSQL):

```
SELECT
  ms.StyleName,
  COUNT(e.EngagementNumber) AS TotalBookings
FROM Engagements AS e
JOIN Entertainer_Styles AS es ON e.EntertainerID = es.EntertainerID
JOIN Musical_Styles AS ms ON es.StyleID = ms.StyleID
WHERE e.CustomerID IN (
    SELECT mp.CustomerID
    FROM Musical_Preferences AS mp
    JOIN Musical_Styles AS ms_pref
      ON mp.StyleID = ms_pref.StyleID
    WHERE ms_pref.StyleName = 'Jazz'
)
GROUP BY ms.StyleName
ORDER BY TotalBookings DESC;
```


musical_styles 1
SELECT ms.StyleName, COUN
Enter a SQL expression to filter results (use Ctrl+Space)
Ctrl+click to open SQL console
Grid
Text
Record
Panels
Refresh
Save
Cancel
Export data
200
17
PST
en
Writable
Smart Insert
22:

	stylename	totalbookings
1	Variety	8
2	Contemporary	8
3	Jazz	8
4	Rhythm and Blues	7
5	Salsa	7
6	Top 40 Hits	6
7	Standards	5
8	Motown	4
9	Country	4
10	Show Tunes	4
11	Country Rock	3

17 row(s) fetched - 0.0s, on 2025-11-14 at 19:26:08



TuneWorks Entertainment Data Dictionary

Agents

Column Name	Data Type	Description
AgentID	serial	Primary key (auto-increment ID) for the agent.
AgtFirstName	varchar(25)	Agent’s first name.
AgtLastName	varchar(25)	Agent’s last name.
AgtStreetAddress	varchar(50)	Agent’s street address.
AgtCity	varchar(30)	Agent’s city.

AgtState	varchar(2)	Agent’s state (2-letter code).
AgtZipCode	varchar(10)	Agent’s ZIP code.
AgtPhoneNumber	varchar(15)	Agent’s phone number.
DateHired	date	Date the agent was hired.
Salary	decimal(15,2)	Agent’s salary (default 0).
CommissionRate	float(24)	Agent’s commission rate (e.g. 0.04 for 4%).

Customers

Column Name	Data Type	Description
CustomerID	serial	Primary key (auto-increment ID) for the customer.
CustFirstName	varchar(25)	Customer’s first name.
CustLastName	varchar(25)	Customer’s last name.
CustStreetAddress	varchar(50)	Customer’s street address.
CustCity	varchar(30)	Customer’s city.
CustState	varchar(2)	Customer’s state (2-letter code).
CustZipCode	varchar(10)	Customer’s ZIP code.

CustPhoneNumber	varchar(15)	Customer’s phone number.

Engagements

Column Name	Data Type	Description
EngagementNumber	serial	Auto-generated unique ID for each engagement (primary key).
StartDate	date	Scheduled start date of the engagement.
EndDate	date	Scheduled end date of the engagement (same as start date if single-day event).
StartTime	time	Scheduled start time of day for the engagement.
StopTime	time	Scheduled end time of day for the engagement.
ContractPrice	decimal(15,2)	Contract price/fee for the engagement (monetary amount, e.g. in USD).
CustomerID	int	ID of the customer who booked this engagement (foreign key to Customers table).
AgentID	int	ID of the agent handling this engagement (foreign key to Agents table).

EntertainerID	int	ID of the entertainer performing (foreign key to Entertainers table).

Entertainer_Members

Column Name	Data Type	Description
EntertainerID	int	Entertainer’s ID (references Entertainers.EntertainerID).
MemberID	int	Member’s ID (references Members.MemberID).
Status	smallint	Membership status code (e.g. active vs. inactive) for the member in the group.

Composite primary key: (EntertainerID, MemberID). Links entertainers to their individual members.

Entertainer_Styles

Column Name	Data Type	Description
EntertainerID	int	Entertainer’s ID (references Entertainers.EntertainerID).
StyleID	int	Music style ID (references Musical_Styles.StyleID).

StyleStrength	smallint	Strength/rating of entertainer’s proficiency in this style (e.g. 1=strongest, 3=weakest).

Composite primary key: (EntertainerID, StyleID). Links entertainers to the musical styles they perform.

Entertainers

Column Name	Data Type	Description
EntertainerID	serial	Primary key (auto-increment ID) for the entertainer.
EntStageName	varchar(50)	Entertainer’s stage or group name.
EntSSN	varchar(12)	Entertainer’s Social Security Number.
EntStreetAddress	varchar(50)	Entertainer’s street address.
EntCity	varchar(30)	Entertainer’s city.
EntState	varchar(2)	Entertainer’s state (2-letter code).
EntZipCode	varchar(10)	Entertainer’s ZIP code.
EntPhoneNumber	varchar(15)	Entertainer’s phone number.
EntWebPage	varchar(50)	Entertainer’s website URL.
EntEMailAddress	varchar(50)	Entertainer’s email address.
DateEntered	date	Date the entertainer was entered/registered.

Members

Column Name	Data Type	Description
MemberID	serial	Primary key (auto-increment ID) for the member.
MbrFirstName	varchar(25)	Member’s first name.
MbrLastName	varchar(25)	Member’s last name.
MbrPhoneNumber	varchar(15)	Member’s phone number.
Gender	varchar(2)	Member’s gender (e.g. 'M' or 'F').

Musical_Preferences

Column Name	Data Type	Description
CustomerID	int	Customer’s ID (references Customers.CustomerID).
StyleID	int	Music style ID (references Musical_Styles.StyleID).
PreferenceSeq	smallint	Customer’s preference order for this style (1 = highest preference).

Composite primary key: (CustomerID, StyleID). Links customers to their preferred music styles.

Musical_Styles

Column Name	Data Type	Description

StyleID	serial	Primary key (auto-increment ID) for the music style.
StyleName	varchar(75)	Name of the musical style (e.g., "Jazz", "Country").

ztblDays

Column Name	Data Type	Description
DateField	date	Single date (likely part of a date dimension table).

ztblMonths

Column Name	Data Type	Description
MonthYear	varchar(15)	Text label for the month and year (e.g., "Jan 2017").
YearNumber	smallint	Numeric year (e.g., 2017).
MonthNumber	smallint	Numeric month (1 = January, etc.).
MonthStart	date	First date of the month.
MonthEnd	date	Last date of the month.
January	smallint	Indicator flag for January (1 if MonthNumber=1, else 0).
February	smallint	Indicator flag for February.
March	smallint	Indicator flag for March.

April	smallint	Indicator flag for April.
May	smallint	Indicator flag for May.
June	smallint	Indicator flag for June.
July	smallint	Indicator flag for July.
August	smallint	Indicator flag for August.
September	smallint	Indicator flag for September.
October	smallint	Indicator flag for October.
November	smallint	Indicator flag for November.
December	smallint	Indicator flag for December.

Likely a month-dimension table. Each row represents one month-year; the January–December columns are flags identifying the month.

ztblSkipLabels

Column Name	Data Type	Description
LabelCount	int	Integer count, possibly used in reporting (purpose unclear).

ztblWeeks

Column Name	Data Type	Description

WeekStart	date	Start date of the week (e.g., Monday).
WeekEnd	date	End date of the week (following Sunday).

Each row represents one calendar week.

Key Table Relationships

- **Engagements–Customers:** Engagements.CustomerID → Customers.CustomerID. Each engagement is booked for a specific customer.
- **Engagements–Agents:** Engagements.AgentID → Agents.AgentID. Each engagement is handled by a specific agent.
- **Engagements–Entertainers:** Engagements.EntertainerID → Entertainers.EntertainerID. Each engagement involves a specific entertainer.
- **Entertainer_Members:** Composite (EntertainerID, MemberID) links Entertainers.EntertainerID to Members.MemberID. (Status indicates active/inactive membership.)
- **Entertainer_Styles:** Composite (EntertainerID, StyleID) links Entertainers.EntertainerID to Musical_Styles.StyleID, with StyleStrength rating.
- **Musical_Preferences:** Composite (CustomerID, StyleID) links Customers.CustomerID to Musical_Styles.StyleID with a preference order.
- **Musical_Styles:** Referenced by Entertainer_Styles and Musical_Preferences.
- **Members:** Referenced by Entertainer_Members.
- **Agents:** Referenced by Engagements.

- **Others (ztbl_*):** Likely supporting date/calendar dimensions (no direct foreign-key links shown).