CS258 Coursework 2025

GigSystem

- Music promotion company is booking acts for a festival.

- Need to store/query information about:
  - acts playing gigs at
  - venues and
  - about tickets.

Act

Act	
actID	SERIAL
actname	VARCHAR(100)
genre	VARCHAR(10)
standardfee	INTEGER

Act names are unique

Genre such as 'rock', 'pop', 'classical'

- Just examples, don't restrict genres

Standard fee is never negative

- Might perform for free

Venue

Venue	
venueid	SERIAL
venuename	VARCHAR(100)
hirecost	INTEGER
capacity	INTEGER

Hirecost in £

- Never negative
- Might be free

Capacity (maximum number of customers allowed at once)

Gig

Gig	
gigid	SERIAL
venueid	INTEGER
gigtitle	VARCHAR(100)
gigdatetime	TIMESTAMP
gigstatus	VARCHAR(10)

Gigs:

- Take place in a venue
- At a particular DateTime
- GigStatus is either "G" or "C" (case-sensitive).

act_gig

act_gig	
actid	INTEGER
gigid	INTEGER
actgigfee	INTEGER
ontime	TIMESTAMP
duration	INTEGER

- Each act charges an actgigfee (in £)
  - May be different to their standardfee
  - Not negative (may be free)
- Start at an ontime
  - At or after the gigdatetime
- Lasts for a duration (in minutes)
- May perform multiple times at a gig

gig_ticket

gig_ticket	
gigID	INTEGER
pricetype	VARCHAR(2)
price	INTEGER

- Ticket prices are not negative (but may be 0 for free tickets)
- Each gig may have different types of ticket, with different prices (in £)
- Every gig will always have a standard ticket type 'A' (for 'Adult')
  - Never free
- Some may also have concessionary tickets, e.g.
  - 'C' for Children
  - 'F' for complementary tickets

Ticket

Ticket	
ticketid	SERIAL
gigID	INTEGER
pricetype	VARCHAR(2)
Cost	INTEGER
CustomerName	VARCHAR(100)
CustomerEmail	VARCHAR(100)

➢ Customer is identifiable by their email address

- No two customers share an email address
- A single customer always uses same name and email address
- Cost of the ticket is the same as the price of the ticket

  Unless the ticket is cancelled, when it becomes 0
- Customers may buy multiple tickets for the same gig
  - Because they might be bringing friends

Business Rules (part 1 of 3)

- There must be no overlap between acts at a gig (although one act can start as soon as the previous act has finished, e.g. it's fine if act 1 finishes at 20:30 and act 2 starts at 20:30)
- Acts cannot perform in multiple gigs at the same time
- Acts can perform multiple times at the same gig (e.g., a first half, then an interval, then a second half)
- Acts only receive one fee per gig, e.g. if their actgigfee is £1000, they only receive £1000 regardless of how many times they perform at a single gig
- Each performance is no shorter than 15 minutes and no longer than 90 minutes
- The same act cannot perform twice without a break (either an interval or a different act playing)

Business Rules (part 2 of 3)

- Acts can perform multiple gigs on the same day but need a 60 minute gap to travel between venues (all venues are close together, and don't require different travelling times)
- Acts can travel to a different gig before the end of a gig (as long as the act has finished performing)
- Venues can be used by multiple gigs on the same day (provided it's not at the same time) but need a 180 minute gap between gigs (so that staff can tidy the venue)
- All intervals (breaks where no act is playing) must be no shorter than 10 minutes and no longer than 30 minutes
- The first act must start at the same ontime as the gigdatetime of the gig

Business Rules (part 3 of 3)

- For each gig, there should not be more tickets sold than the capacity of the venue
- The final act of a gig must finish at least 60 minutes after the start of the gig
- Gigs involving genres 'Rock' or 'Pop' (case sensitive) at any point in the gig must finish no later than 11pm (inclusive) to prevent disruption to neighbouring residents, all other gigs must finish by 1am
- Gigs cannot start after 11:59pm. The earliest a gig can start is 9am
- All times can be assumed to be rounded to the nearest minute
- Venue/Act clashes can ignore cancelled gigs

Part 1: The Schema

- Write a schema.sql file to create the necessary tables
- Use SQL to specify any key and referential integrity constraints
- Capture the system requirements as accurately as possible
- Also specify any additional constraints that you believe should apply
- Your schema will also contain any extra DDL statements that you use for the application
- These are marked with the task methods you implement

Part 2: Design Choices

- You're not allowed to change the database structure
- Must reflect the tables provided in specification
- But if you could change the tables, what would you change?
- Write your answer in a Design Choices section of README.md

Part 3: The Application

- We give you:
  - GigSystem.java (a bit like BioSystem.java)
  - GigTester.java a basic testing suite (some tests implemented, mostly not)
- Runs tests similar to how we will mark your program

The Tasks

Task 1: Get the line-up

- Find the line-up for any given gigID
- Must be returned as 2D String Array:
  - [[“ViewBee 40”, ”18:00”, ”18:50”], [“The Where”, ”19:00”, ”20:25”], [“The Selecter”, ”20:25”, ”21:25”]]
- But conceptually it gives you this:

Act Name	On Time	Off Time
ViewBee 40	18:00	18:50
The Where	19:00	20:25
The Selecter	20:25	21:25

- Can use the printTable method to help you visualise it for your testing purposes

Task 2: Create a new Gig

- Create a new gig
- Given:
  - String venue, String gigTitle, LocalDateTime gigStart, int adultTicketPrice, *ActPerformanceDetails*[] actDetails
  - *ActPerformanceDetails* contains int actID, int fee; LocalDateTime onTime; int duration;
- If any details of the gig (or acts) violate any constraints
  - ensure the database state is as it was before the method was called

Task 3: Buy a ticket

- A customer wants to buy a ticket
- Given:
  - int gigid, String name, String email, String ticketType
- If any details are inconsistent (e.g. ticket type doesn't match a pricetype)
  - e.g. if the gig does not exist, or there is no matching pricetype, or there is some other error
  - do not allow the ticket purchase and ensure the database state is as it was before the method was called

Task 4: Cancelling an Act

- An act needs to cancel a gig (gigID and actName supplied)
- Remove all performances of the specified act from the specified gig
- Adjust the ontime of all subsequent performances to remove the gap
  - (e.g., if the cancelled act lasts 40 minutes, move all subsequent acts 40 minutes earlier)
- Return the updated line-up (as with Task 1)
- If the cancellation of the act would violate the constraints in the specification
  - or if the act is the headline act (the final or only act)
    - Cancel the entire gig

Task 4: Cancelling an Act

- If an entire gig is cancelled
  - Leave all gig line-up (act_gig) details as they are (do not remove the cancelling act)
  - change the cost of all tickets sold for that gig to be 0 (but do not change the original price of the ticket)
  - Return an array of strings representing names and email addresses of customers who have affected tickets, ordered by customer name (ascending alphabetical order), containing no duplicates.

Task 5: Tickets Needed to Sell

- For each gig, find how many more tickets (of the cheapest ticket price for that gig) still need to be sold
- promoters need to be able to pay
  - agreed act fees (as listed in act_gig)
  - venue fee
- Ordered by gigid (smallest first), include those that haven't sold any tickets yet
  - Return 2D string array [[1,1600],[2,2000],[3,1525]]
  - 3 rows <1, 1600>, <2, 2000>, <3, 1525>

Task 6: How Many Tickets Sold

- Total number of tickets each act has sold
  - Any pricetype
  - When they were a headline act
  - Not including cancelled gigs
- For each act, show the number per year and the total number
- Order with those that have sold least first, then order by year (with total at the end of each group)

Act Name	Year	Total Tickets Sold
UB40	2018	5,000
UB40	2019	7,500
UB40	**TOTAL**	**12,500**
The Selecter	2020	4,000
The Selecter	2021	6,000
The Selecter	**TOTAL**	**10,000**

Task 7: Regular Customers

- Want to know who regularly attends gigs that feature particular acts
- Each act who has ever performed a gig as a headline act
- With names of customers who have attended at least one of these gigs per calendar year
    - If the act performed such a gig as headline act in that year
- Output 2D String Array, sorted by:
    - Acts in Alphabetical Order
    - Customers in order of number of tickets the customer has bought for a gig where that act was the headline act
- Not including cancelled gigs
- If no such customers, and the act has played as a headline act, keep act listed but put '[None]' in the customer column

Act	Customer
Join Division	G Jones
Join Division	Jonny F
Fontaine D.C.	Peter T.
Fontaine D.C.	Theo T.
The National	Eleni T.
The National	Peter T.

Task 8: Economically Feasible Gigs

- Want to organise a gig with a single act

  Only want to charge average ticket price (excluding cancelled gigs)
- Trying to find acts that would suit venues
  - Need to pay for the venue and the act's standard fee
- Need to know number of tickets that would need to be sold
  - Order in alphabetical order of venue name, then tickets required (highest first)

Writing Your Code

- Must not modify method signatures in GigSystem.java
- Use existing convertResultToStrings method and printTable method.
- For GigSystem.java, you must only import what is immediately available on the DCS system (do not use external libraries, and do not modify the Maven pom.xml file).
- Other than parts of java.sql, we strongly recommend that you do not import other libraries (the best solutions will only require minimal Java).
  - For GigTester.java, you can import any Java library, as you don't need to submit it.
- Full instructions on getting started can be found in INSTRUCTIONS.md

Testing and Marking

- Must maintain database integrity
  - Must not violate business rules
- Before Java methods called, DB in a known good state (with test data)
- Java methods must ensure DB is in a consistent state afterwards
  - Java methods called (maybe with inconsistent information)
  - After method called, we check integrity
- See specification document for information on TRIGGERs
- Only method outputs are tested - no need for "friendly" error messages
  - better to show full stack trace to help markers
- Task methods independent (not called in order)
- Do not use schema paths (do not use CREATE SCHEMA)

Testing and Marking

- Test Suite provided
  - Contains some test cases for some tasks
  - You are advised to write your own
    - At least ensure correct output style
  - If it doesn't work in the test suite, likely to score 0
  - You cannot submit your GigTester.java
- Sample Data — does not break business rules
  - testbig.sql — for most parts
  - testsmall.sql — to help with tasks 7 and 8
- Script to generate some random data provided
  - Random data may break rules — designed to help you find inconsistencies!
- Serial sequences set at 10,001 — test data will use lower values

Submission

Submit three files

- schema.sql — all necessary SQL definitions (not just tables)
- README.md — 'Design Choices' and explanations of solutions
  - Each task should have 50-300 words explaining how your solution works
  - Any Java (other than short JDBC communication) should be commented
- GigSystem.java — completed Java file

Best Solutions Use Minimal Java

- Use prepared statements (for security and efficiency)
- Handle the unexpected (so database is not inconsistent)
  - But we only mark the returned outputs
- Use SQL where possible, not procedural code
  - Honestly, SQL is much better at Ordering/Filtering
  - Should be no need to restructure data, can use convertResultToStrings
- Must be compatible with DCS servers (Postgres 16)
- Must not collaborate with others
- NB: Plagiarism and Use of LLMs

Questions?

- The module page will contain a FAQ section/link
- Any and all questions should be submitted via email **to all**:
  - Chung Underwood, Faye
    - <Faye.Chung-Underwood.1@warwick.ac.uk>;
  - ZHANG, MIN (PGR
    - <Min.Zhang.2@warwick.ac.uk>;
  - ATAEE, DANIAL (PGR)
    - <Danial.Ataee@warwick.ac.uk>;
  - YAN, DA (PGR)
    - <Da.Yan@warwick.ac.uk>