

Session 14

Change control applied to the Session 13 RRS, from demonstration to CCB decision to rrs_v2.py

This session takes the `rrs.py` baseline from Session 13, shows it to another person, records every change they ask for, and then implements one of them the way a maintained product does it: a numbered change request, an impact analysis, a decision by a change control board, an implementation in a new version, verification, a release note, and a second demonstration to the same person. Most software cost is spent after the first release, so the skill being assessed here is not coding the change. It is proving that the change was chosen, bounded and checked before the code moved. The deliverable is the paper trail plus `rrs_v2.py` and a diff against the baseline.

Objectives

✗ Do not copy. Read for understanding and the viva

- Run a structured demonstration and capture suggestions as change requests with priority and effort, not as a wish list
- Fill a change request form and an impact analysis that names every artefact from Sessions 1 to 13 the change touches
- Record a change control board decision with reasons for accepting one request and deferring the rest
- Implement the accepted change in a new version, keeping the baseline untouched so the diff is reviewable
- Verify the change, regression-test the old behaviour, write the release note, and get sign-off from the same reviewer

Problem Statement

■ Write in lab record

Demonstrate the Software developed in Session 13 to any other person and make a list of Changes suggested by him/her. Implement changes in Software following the Change Control process and demonstrate the updated Software to the same person.

Concept

✖ Do not copy. Read for understanding and the viva

Why a process and not just an edit

A request that arrives as “can you add Tatkal?” hides three decisions: is it in scope, what else does it break, and is it worth doing now. Change control makes each decision visible and signed. The baseline (`rrs.py`), the Session 3 SRS, the Session 4 data dictionary, the Session 13 traceability matrix) is frozen; every change is a numbered request against it.

The steps applied in this session

1. Change request raised: who, what, why, priority, estimated effort.
2. Impact analysis: which requirements, data, code, tests and documents change.
3. Change control board (CCB) decision: accept, defer or reject, with a reason.
4. Implementation in a new version identifier. The baseline file is not edited.
5. Verification: new tests for the change, old tests re-run for regression.
6. Release note and re-demonstration; the requester signs off.

Impact analysis is the step students skip

Ask, for the change: which FR ids change or are added, which entity attributes change, which functions change, which existing tests could break, which pages of the record need an update. If the answer to the last question is “none”, the analysis is incomplete, because at least the traceability matrix gains a row.

Version identification

`rrs.py` is version 1.0, the baseline. `rrs_v2.py` is version 1.1, one accepted change. Keeping both files means the diff between them is the implementation record. In a real project this is a commit on a branch; in the lab record it is a copy with a new name.

Demonstration Record

■ Write in lab record

Item	Value
Software	RRS version 1.0, <code>rrs.py</code> from Session 13
Demonstrated to	Nikhil Sharma, MCA second year, same study centre, has used IRCTC as a passenger
Demonstrated by	Student (author of the Session 13 build)
Date and duration	2026-09-20, 40 minutes
Environment	macOS 27.0, Python 3.13.5, terminal, fresh <code>rrs_data.json</code>
Scenario shown	Admin adds a schedule; passenger searches BCT to NDLS, books 2 passengers in 2A, books until 1A is waitlisted, looks up PNR, cancels and sees the waitlist promoted
Reviewer was asked	What is missing, what is confusing, what would you change first

Suggested Changes

■ Write in lab record

Priority: H = blocks real use, M = useful, L = cosmetic. Effort estimated in hours against the Session 2 baseline of about 60 LOC per hour for this code base.

CR id	Suggestion by reviewer	Priority	Effort	Type
CR-01	Show which berth was allotted (LB, UB) and honour the berth preference	M	3 h	Enhancement
CR-02	Availability should be per segment; a BCT to KOTA booking should not block KOTA to NDLS	H	8 h	Defect against SRS intent
CR-03	Tatkal quota: reserve 10% of seats per class, bookable only the day before departure, at a 30% premium	H	3 h	New requirement
CR-04	Passenger login so a person sees only their own PNRs	M	5 h	Deferred module (FR-1.1 to FR-1.6)
CR-05	Occupancy report per schedule for the administrator	L	2 h	Deferred module 8
CR-06	Print the cancellation confirmation with the refund amount in the PNR lookup	L	1 h	Enhancement

Change Request Form

■ Write in lab record

Field	Entry
CR id	CR-03
Title	Tatkal quota with day-before booking window and fare premium
Raised by	Nikhil Sharma (reviewer), 2026-09-20
Raised against	RRS 1.0, <code>rrs.py</code> , SRS Session 3
Description	Reserve 10% of seats in each coach class (rounded up) as a Tatkal pool. A Tatkal booking is accepted only when the booking date is exactly one day before the run date. Fare is the normal fare plus 30%. Tatkal bookings are not waitlisted; if the pool is full the booking is refused. General bookings never use the Tatkal pool
Reason	Every real Indian train has a Tatkal quota; the reviewer said the demo felt incomplete without it. It is a real requirement the SRS missed
Priority	High
Estimated effort	3 hours: 1 design and impact analysis, 1 code, 1 test and record
Requested for	Version 1.1
Status	Proposed on 2026-09-20, Accepted on 2026-09-22, Implemented on 2026-09-25, Verified on 2026-09-26

Impact Analysis

■ Write in lab record

Artefact	Impact of CR-03	Change needed
Session 1 scope	Booking module gains a quota concept	One sentence added to the scope of Booking
Session 3 SRS	New requirements	Add FR-4.10 (Tatkal pool per class), FR-4.11 (day-before window), FR-4.12 (30% premium, no waitlist). Business rule list gains the 10% figure
Session 4 data dictionary	Booking entity	Add attribute <code>quota</code> with values G or T. Coach and Seat unchanged: the pool is the last 10% of labels, derived, not stored
Session 4 DFD	Process 4.2 Allocate Seats and 4.3 Compute Fare	Both read quota; no new process or store
Session 5 design	Booking module interface	Input gains quota; Cancellation module unchanged because promotion already uses the general pool only
Session 6 screens	Booking form and Availability screen	One new prompt (Quota) and one new column (Tatkal free count)
Session 13 code	<code>free_seats</code> , <code>print_availability</code> , <code>book_ticket</code> , constants, imports	9 hunks, 28 lines added, 11 removed. <code>cancel_ticket</code> , <code>promote_waitlist</code> , <code>add_train</code> , <code>add_schedule</code> , search unchanged
Session 13 tests	Validation checklist	All 12 rows re-run. Rows for seat order and waitlist now assume the class has more than 10 seats
Session 13 traceability	New rows FR-4.10 to FR-4.12	Map to <code>free_seats</code> , <code>book_ticket</code>
Data file	<code>rrs_data.json</code>	Old bookings have no <code>quota</code> key; nothing reads it back, so old files still load

Artefact	Impact of CR-03	Change needed
Side effect found	A class with fewer than 10 seats loses $\text{ceil}(10\%) = 1$ seat to Tatkal	The 1-seat Demo Express class from Session 13 has 0 general seats in 1.1; the seed trains (18 to 216 seats per class) are unaffected. Recorded as a known behaviour, not a defect, since real coaches have 18 or more seats
Risk	Date comparison uses the calendar day, not a clock time	Tatkal opens at 00:00 the day before, not at 10:00 as on IRCTC. Accepted for 1.1

CCB Decision

■ Write in lab record

CCB for a lab project: the student (developer), the reviewer (customer), and one more student acting as the quality member. Met on 2026-09-22.

CR id	Decision	Reason
CR-03	Accepted for 1.1	High priority, three hours, touches one module, does not change any existing rule. Reviewer ranked it first
CR-02	Deferred to 1.2	High priority but eight hours and changes the seat model in <code>free_seats</code> for every caller; must be done alone, with a redesign of the allocation table in Session 4
CR-04	Deferred to 1.2	Reinstates module 1, which the Session 13 build cut on purpose; needs the User entity and password hashing, five hours
CR-01	Deferred	Depends on CR-02: berth allocation only makes sense once seats are tracked per segment
CR-05	Deferred	Low priority; the data is already readable from the JSON file
CR-06	Rejected	The refund is already visible at cancellation time; the lookup line shows status CANCELLED. Not worth a version

One change per version is the rule applied: a single-change diff can be reviewed and reverted; a bundle cannot.

Implementation

■ Write in lab record

Version 1.1 is `rrs_v2.py`, a copy of `rrs.py` with only CR-03 applied. Two constants carry the numbers from the change request so they are not scattered in the code.

rrs_v2.py

```

1  #!/usr/bin/env python3
2  """Railway Reservation System (RRS) - MCS-217 Session 14 (v2, CR-03 Tatkal q
3
4  Console implementation of modules 2 to 5 of the RRS specified in
5  Sessions 1, 3, 4, 5 and 6:
6      2. Train and Schedule Management    (Administrator)
7      3. Search and Availability          (Passenger)
8      4. Booking                          (Passenger)
9      5. Cancellation and Refund          (Passenger)
10
11  Standard library only. All data lives in rrs_data.json next to this file.
12  """
13  import json
14  import math
15  import os
16  import random
17  from datetime import date, datetime, timedelta
18
19  DATA_FILE = os.path.join(os.path.dirname(os.path.abspath(__file__)), "rrs_da
20  CLASSES = ("SL", "3A", "2A", "1A")
21  BERTHS = ("LB", "MB", "UB", "SL", "SU")
22  CLERKAGE = 60          # flat clerkage in rupees (business rule, Session 3)
23  MAX_PASSENGERS = 6     # per PNR (business rule, Session 3)
24  TATKAL_SHARE = 0.10    # CR-03: seats per class reserved for Tatkal
25  TATKAL_PREMIUM = 0.30  # CR-03: fare premium on Tatkal bookings
26
27  # ----- persisten
28
29
30  def load_data():
31      """Read rrs_data.json; seed two trains and two schedules on first run."""
32      if os.path.exists(DATA_FILE):
33          with open(DATA_FILE) as fh:
34              return json.load(fh)
35      data = {"trains": {}, "schedules": {}, "bookings": {}}
36      data["trains"]["12951"] = {
37          "name": "Mumbai Rajdhani", "type": "Rajdhani",
38          "route": [
39              {"station_code": "BCT", "name": "Mumbai Central", "sequence": 1,
40               "arrival_time": "--", "departure_time": "17:00", "distance_km":
41               {"station_code": "KOTA", "name": "Kota Jn", "sequence": 2,
42                "arrival_time": "01:35", "departure_time": "01:40", "distance_k
43               {"station_code": "NDLS", "name": "New Delhi", "sequence": 3,

```

```

44         "arrival_time": "08:35", "departure_time": "--", "distance_km":
45     ],
46     "coaches": {"3A": [2, 64], "2A": [1, 48], "1A": [1, 18]},
47     "fares": {"3A": 2.10, "2A": 2.90, "1A": 4.80},
48 }
49 data["trains"]["12137"] = {
50     "name": "Punjab Mail", "type": "Mail",
51     "route": [
52         {"station_code": "CSMT", "name": "Mumbai CSMT", "sequence": 1,
53          "arrival_time": "--", "departure_time": "19:35", "distance_km":
54          {"station_code": "BPL", "name": "Bhopal Jn", "sequence": 2,
55           "arrival_time": "09:15", "departure_time": "09:25", "distance_k
56           {"station_code": "NDLS", "name": "New Delhi", "sequence": 3,
57            "arrival_time": "20:35", "departure_time": "--", "distance_km":
58         ],
59     "coaches": {"SL": [3, 72], "3A": [1, 64]},
60     "fares": {"SL": 0.60, "3A": 1.60},
61 }
62 in_10_days = (date.today() + timedelta(days=10)).isoformat()
63 for sid, tno in (("S001", "12951"), ("S002", "12137")):
64     data["schedules"][sid] = new_schedule(tno, in_10_days, data["trains"]
65 save_data(data)
66 return data
67
68
69 def save_data(data):
70     """Write the whole data dictionary back to rrs_data.json."""
71     with open(DATA_FILE, "w") as fh:
72         json.dump(data, fh, indent=1)
73
74
75 def new_schedule(train_no, run_date, train):
76     """Build a Schedule record with empty seat maps and waitlists per class.
77     return {"train_no": train_no, "run_date": run_date, "status": "SCHEDULED
78             "allocated": {c: {} for c in train["coaches"]},
79             "waitlist": {c: [] for c in train["coaches"]}}
80
81 # ----- input hel
82
83
84 def ask(prompt, check=lambda s: True, error="Invalid value, try again."):
85     """Prompt until check(value) is true; returns the stripped string."""
86     while True:
87         value = input(prompt).strip()

```

```

88         if value and check(value):
89             return value
90         print(error)
91
92
93 def ask_int(prompt, lo, hi):
94     """Prompt for an integer in the closed range lo..hi."""
95     return int(ask(prompt, lambda s: s.isdigit() and lo ≤ int(s) ≤ hi,
96                     "Enter a number from %d to %d." % (lo, hi)))
97
98
99 def ask_date(prompt):
100     """Prompt for a date in YYYY-MM-DD form."""
101
102     def ok(s):
103
104         try:
105
106             date.fromisoformat(s)
107
108             return True
109
110         except ValueError:
111
112             return False
113
114     return ask(prompt, ok, "Use the form YYYY-MM-DD.")
115
116
117 # ----- domain he
118
119
120
121
122
123 def seat_labels(train, cls):
124
125     """All seat labels of a class in allocation order, e.g. 3A1-17."""
126
127     coaches, per_coach = train["coaches"][cls]
128
129     return ["%s%d-%d" % (cls, c, s) for c in range(1, coaches + 1)

```

```

11
6         for s in range(1, per_coach + 1)]
11
7
11
8
11
9 def free_seats(data, sched, cls, quota="G"):
12
0     """Unallocated seat labels of this class in the General or Tatkal pool.
12
1
12
2     CR-03: the last 10% of each class (rounded up) is the Tatkal pool.
12
3     """
12
4     labels = seat_labels(data["trains"][sched["train_no"]], cls)
12
5     cut = len(labels) - math.ceil(len(labels) * TATKAL_SHARE)
12
6     pool = labels[cut:] if quota == "T" else labels[:cut]
12
7     return [s for s in pool if s not in sched["allocated"][cls]]
12
8
12
9
13
0 def departure(data, sched):
13
1     """Datetime at which the schedule leaves its first station."""
13
2     first = data["trains"][sched["train_no"]]["route"][0]
13
3     return datetime.fromisoformat(sched["run_date"] + " " + first["departure
13
4
13
5
13
6 def stop_index(train, code):
13
7     """Index of a station code in the train route, or -1."""

```

14 / 34

```
16
0     return found
16
1
16
2
16
3 def print_availability(data, sched):
16
4     """One line per class: free seats or waitlist length."""
16
5     for cls in sched["allocated"]:
16
6         free, wl = len(free_seats(data, sched, cls)), len(sched["waitlist"][
16
7         tatkal = len(free_seats(data, sched, cls, "T"))
16
8         print("    %-3s %s    Tatkal %d" % (cls, "AVAILABLE %d" % free if free
16
9
17
0
17
1 def promote_waitlist(data, sched, cls):
17
2     """Move waitlisted bookings of this class to CONFIRMED while seats allow
17
3     wl = sched["waitlist"][cls]
17
4     while wl:
17
5         booking = data["bookings"][wl[0]]
17
6         free = free_seats(data, sched, cls)
17
7         if len(free) < len(booking["passengers"]):
17
8             return
17
9         for p, seat in zip(booking["passengers"], free):
18
0             p["seat"] = seat
18
1             sched["allocated"][cls][seat] = wl[0]
```

```

18
2     booking["status"] = "CONFIRMED"
18
3     print("Waitlisted PNR %s promoted to CONFIRMED." % wl.pop(0))
18
4
18
5 # ----- admin act
18
6
18
7
18
8 def add_train(data):
18
9     """Admin: add a train with its route stations, coaches and per-km fares.
19
0     train_no = ask("Train number (5 digits): ", lambda s: s.isdigit() and le
19
1     if train_no in data["trains"]:
19
2         print("Train %s already exists." % train_no)
19
3         return
19
4     train = {"name": ask("Train name: "), "type": ask("Type (Rajdhani/Mail/E
19
5         "route": [], "coaches": {}, "fares": {})}
19
6     stops = ask_int("Number of stations on the route (2-20): ", 2, 20)
19
7     for seq in range(1, stops + 1):
19
8         print("Station %d of %d" % (seq, stops))
19
9         train["route"].append({
20
0             "station_code": ask("  code: ").upper(), "name": ask("  name: ")
20
1             "arrival_time": "--" if seq == 1 else ask("  arrival HH:MM: "),
20
2             "departure_time": "--" if seq == stops else ask("  departure HH:
20
3             "distance_km": 0 if seq == 1 else ask_int("  distance from origi

```

```

20
4     for cls in CLASSES:
20
5         n = ask_int("Coaches of class %s (0-10): " % cls, 0, 10)
20
6         if n:
20
7             train["coaches"][cls] = [n, ask_int("  seats per %s coach (1-80)
20
8             train["fares"][cls] = float(ask("  fare per km for %s (rupees):
20
9                                     lambda s: s.replace(".", "", 1).
21
0     data["trains"][train_no] = train
21
1     save_data(data)
21
2     print("Train %s %s added with %d stations." % (train_no, train["name"],
21
3
4
21
5 def add_schedule(data):
21
6     """Admin: create a run of an existing train on a given date."""
21
7     list_trains(data)
21
8     train_no = ask("Train number: ", lambda s: s in data["trains"], "No such
21
9     run_date = ask_date("Run date (YYYY-MM-DD): ")
22
0     if any(s["train_no"] == train_no and s["run_date"] == run_date for s in
22
1         print("That train already runs on %s." % run_date)
22
2         return
22
3     sid = "S%03d" % (len(data["schedules"]) + 1)
22
4     data["schedules"][sid] = new_schedule(train_no, run_date, data["trains"]
22
5     save_data(data)

```

```

22
6     print("Schedule %s created: %s on %s." % (sid, train_no, run_date))
22
7
22
8
22
9 def list_trains(data):
23
0     """Admin: print every train with its route and classes."""
23
1     for train_no, t in sorted(data["trains"].items()):
23
2         stations = " > ".join(s["station_code"] for s in t["route"])
23
3         print("%s %-16s %s classes: %s" % (train_no, t["name"], stations, "
23
4
23
5 # ----- passenger
23
6
23
7
23
8 def search_trains(data):
23
9     """Passenger: list trains between two stations on a date with availabili
24
0     src, dst = ask("From station code: ").upper(), ask("To station code: ").
24
1     run_date = ask_date("Journey date (YYYY-MM-DD): ")
24
2     found = find_schedules(data, src, dst, run_date)
24
3     if not found:
24
4         print("No trains found for %s to %s on %s." % (src, dst, run_date))
24
5         return
24
6     for sid, sched, train, i, j in found:
24
7         km = train["route"][j]["distance_km"] - train["route"][i]["distance_

```

```

24
8     print("%s %s %s dep %s arr %s %d km" % (sid, sched["train_no"],
24
9         train["route"][i]["departure_time"], train["route"][j]["arrival_time"],
25
0     print_availability(data, sched)
25
1
25
2
25
3 def check_availability(data):
25
4     """Passenger: seat availability per class for one schedule."""
25
5     sid = ask("Schedule id: ", lambda s: s in data["schedules"], "No such schedule")
25
6     sched = data["schedules"][sid]
25
7     print("%s %s on %s" % (sched["train_no"], data["trains"][sched["train_no"]],
25
8     print_availability(data, sched)
25
9
26
0
26
1 def book_ticket(data):
26
2     """Passenger: book up to six passengers, allocate seats, generate PNR."""
26
3     sid = ask("Schedule id: ", lambda s: s in data["schedules"], "No such schedule")
26
4     sched = data["schedules"][sid]
26
5     train = data["trains"][sched["train_no"]]
26
6     if departure(data, sched) <= datetime.now():
26
7         print("This train has already departed. Booking refused.")
26
8         return
26
9     src = ask("From station code: ", lambda s: stop_index(train, s.upper()))

```

```

27
0     dst = ask("To station code: ",
27
1         lambda s: stop_index(train, s.upper()) > stop_index(train, src
27
2     cls = ask("Class (%s): " % "/".join(train["coaches"]), lambda s: s.upper
27
3     quota = ask("Quota (G=General/T=Tatkal): ", lambda s: s.upper() in "GT")
27
4     if quota == "T" and date.today() != date.fromisoformat(sched["run_date"]
27
5         print("Tatkal opens only on the day before departure. Booking refuse
27
6         return
27
7     n = ask_int("Number of passengers (1-%d): " % MAX_PASSENGERS, 1, MAX_PAS
27
8     passengers = []
27
9     for k in range(1, n + 1):
28
10         passengers.append({"name": ask("Passenger %d name: " % k),
28
11                             "age": ask_int("Passenger %d age: " % k, 1, 120),
28
12                             "gender": ask("Passenger %d gender (M/F/O): " % k
28
13                             "berth_preference": ask("Passenger %d berth prefe
28
14
15                             lambda s: s.upper() in BE
28
16     km = train["route"][stop_index(train, dst)]["distance_km"] - train["rout
28
17     premium = 1 + TATKAL_PREMIUM if quota == "T" else 1
28
18     fare = round(train["fares"][cls] * km * n * premium, 2)
28
19     free = free_seats(data, sched, cls, quota)
28
20     if quota == "T" and len(free) < n:
29
21         print("No Tatkal seats left in %s. Booking refused." % cls)
29
22     return

```

```

29 status = "CONFIRMED" if len(free) ≥ n else "WAITLISTED"
29
3 print("Fare: %d km x Rs %.2f/km x %d passengers x %.2f = Rs %.2f  [%s]" %
29
4       km, train["fares"][cls], n, premium, fare, status))
29
5 if ask("Confirm booking (Y/N): ", lambda s: s.upper() in "YN").upper() != "Y":
29
6     print("Booking abandoned.")
29
7     return
29
8 pnr = new_pnr(data)
29
9 if status == "CONFIRMED":
30
0     for p, seat in zip(passengers, free):
30
1         p["seat"] = seat
30
2         sched["allocated"][cls][seat] = pnr
30
3 else:
30
4     sched["waitlist"][cls].append(pnr)
30
5 data["bookings"][pnr] = {"pnr": pnr, "schedule_id": sid, "from_station":
30
6                             "class": cls, "quota": quota, "booking_time": d
30
7                             "status": status, "total_fare": fare, "passenge
30
8 save_data(data)
30
9 print("Booked. PNR %s  status %s" % (pnr, status))
31
0 print_booking(data, pnr)
31
1
2
31
3 def cancel_ticket(data):

```

```
31
4     """Passenger: cancel by PNR, compute refund, promote waitlist."""
31
5     pnr = ask("PNR: ", lambda s: s in data["bookings"], "No such PNR.")
31
6     booking = data["bookings"][pnr]
31
7     if booking["status"] == "CANCELLED":
31
8         print("PNR %s is already cancelled." % pnr)
31
9         return
32
0     sched = data["schedules"][booking["schedule_id"]]
32
1     hours = (departure(data, sched) - datetime.now()).total_seconds() / 3600
32
2     if booking["status"] == "WAITLISTED":
32
3         refund, rule = booking["total_fare"] - CLERKAGE, "waitlisted, fare m
32
4         sched["waitlist"][booking["class"]].remove(pnr)
32
5     elif hours > 48:
32
6         refund, rule = booking["total_fare"] - CLERKAGE, "more than 48 h bef
32
7     elif hours ≥ 12:
32
8         refund, rule = booking["total_fare"] * 0.5, "between 48 h and 12 h b
32
9     else:
33
0         refund, rule = 0.0, "less than 12 h before departure"
33
1     refund = round(max(refund, 0.0), 2)
33
2     print("Refund: Rs %.2f (%s)" % (refund, rule))
33
3     if ask("Confirm cancellation (Y/N): ", lambda s: s.upper() in "YN").uppe
33
4         print("Cancellation abandoned.")
33
5     return
```

```

33
6     if booking["status"] == "CONFIRMED":
33
7         for p in booking["passengers"]:
33
8             del sched["allocated"][booking["class"]][p["seat"]]
33
9             p["seat"] = None
34
0     booking["status"] = "CANCELLED"
34
1     booking["refund"] = {"amount": refund, "reason": rule,
34
2                           "processed_at": datetime.now().isoformat(timespec="
34
3     promote_waitlist(data, sched, booking["class"])
34
4     save_data(data)
34
5     print("PNR %s cancelled." % pnr)
34
6
34
7
34
8 def pnr_lookup(data):
34
9     """Passenger: show a booking by PNR."""
35
0     print_booking(data, ask("PNR: ", lambda s: s in data["bookings"], "No su
35
1
35
2
35
3 def print_booking(data, pnr):
35
4     """Print one booking with its passengers and seats."""
35
5     b = data["bookings"][pnr]
35
6     sched = data["schedules"][b["schedule_id"]]
35
7     print("PNR %s %s %s %s %s > %s class %s fare Rs %.2f %s" % (

```

```

35
8     pnr, sched["train_no"], data["trains"][sched["train_no"]]["name"], s
35
9     b["from_station"], b["to_station"], b["class"], b["total_fare"], b["
36
0     for p in b["passengers"]:
36
1         print("    %-12s %3d %s    pref %-2s    seat %s" % (p["name"], p["age"],
36
2                                                     p["berth_preference"]
36
3
36
4 # ----- menus
36
5
36
6
36
7 def run_menu(title, actions, data):
36
8     """Print a numbered menu and dispatch until the user chooses 0."""
36
9     while True:
37
0         print("\n= %s =" % title)
37
1         for k, (label, _) in enumerate(actions, 1):
37
2             print(" %d. %s" % (k, label))
37
3         print(" 0. Back")
37
4         choice = ask_int("Choice: ", 0, len(actions))
37
5         if choice == 0:
37
6             return
37
7         actions[choice - 1][1](data)
37
8
37
9

```

```
38
0  def main():
38
1      """Entry point: role selection."""
38
2      data = load_data()
38
3      admin = [("Add train", add_train), ("Add schedule", add_schedule), ("Lis
38
4      passenger = [("Search trains", search_trains), ("Check availability", ch
38
5                  ("Book ticket", book_ticket), ("Cancel ticket", cancel_tick
38
6      roles = [("Administrator", lambda d: run_menu("Administrator", admin, d)
38
7                  ("Passenger", lambda d: run_menu("Passenger", passenger, d)))]
38
8      print("Railway Reservation System (Session 14 build, v2)")
38
9      try:
39
0          run_menu("Main menu", roles, data)
39
1      except EOFError:
39
2          pass
39
3      print("Bye.")
39
4
39
5
39
6  if __name__ == "__main__":
39
7      main()
```

Diff Summary

Output of `diff -u session-13/rrs.py session-14/rrs_v2.py`, changed hunks only. Nine hunks, 28 lines added, 11 removed. Everything else in the file is byte-identical.

```

1 @@ -1,5 +1,5 @@
2 -"""Railway Reservation System (RRS) - MCS-217 Session 13.
3 +"""Railway Reservation System (RRS) - MCS-217 Session 14 (v2, CR-03 Tatkal
4 @@ -11,6 +11,7 @@
5 import json
6 +import math
7 import os
8 @@ -20,6 +21,8 @@
9 MAX_PASSENGERS = 6      # per PNR (business rule, Session 3)
10 +TATKAL_SHARE = 0.10    # CR-03: seats per class reserved for Tatkal
11 +TATKAL_PREMIUM = 0.30  # CR-03: fare premium on Tatkal bookings
12 @@ -113,12 +116,17 @@
13 -def free_seats(data, sched, cls):
14 -     """Seat labels of this class not yet allocated on this schedule."""
15 -     train = data["trains"][sched["train_no"]]
16 -     return [s for s in seat_labels(train, cls) if s not in sched["allocated"]
17 +def free_seats(data, sched, cls, quota="G"):
18 +     """Unallocated seat labels of this class in the General or Tatkal pool.
19 +
20 +     CR-03: the last 10% of each class (rounded up) is the Tatkal pool.
21 +
22 +     labels = seat_labels(data["trains"][sched["train_no"]], cls)
23 +     cut = len(labels) - math.ceil(len(labels) * TATKAL_SHARE)
24 +     pool = labels[cut:] if quota == "T" else labels[:cut]
25 +     return [s for s in pool if s not in sched["allocated"][cls]]
26 @@ -156,7 +164,8 @@
27         free, wl = len(free_seats(data, sched, cls)), len(sched["waitlist"])
28 -     print("    %-3s %s" % (cls, "AVAILABLE %d" % free if free else "WL %
29 +     tatkal = len(free_seats(data, sched, cls, "T"))
30 +     print("    %-3s %s Tatkal %d" % (cls, "AVAILABLE %d" % free if free
31 @@ -261,6 +270,10 @@
32     cls = ask("Class (%s): " % "/".join(train["coaches"]), lambda s: s.upper
33 +     quota = ask("Quota (G=General/T=Tatkal): ", lambda s: s.upper() in "GT"
34 +     if quota == "T" and date.today() != date.fromisoformat(sched["run_date"]
35 +         print("Tatkal opens only on the day before departure. Booking refus
36 +         return
37     n = ask_int("Number of passengers (1-%d): " % MAX_PASSENGERS, 1, MAX_PA
38 @@ -270,10 +283,15 @@
39 -     fare = round(train["fares"][cls] * km * n, 2)
40 -     free = free_seats(data, sched, cls)
41 +     premium = 1 + TATKAL_PREMIUM if quota == "T" else 1
42 +     fare = round(train["fares"][cls] * km * n * premium, 2)
43 +     free = free_seats(data, sched, cls, quota)

```

```
44 +     if quota == "T" and len(free) < n:
45 +         print("No Tatkal seats left in %s. Booking refused." % cls)
46 +         return
47     status = "CONFIRMED" if len(free) ≥ n else "WAITLISTED"
48 -     print("Fare: %d km x Rs %.2f/km x %d passengers = Rs %.2f  [%s]" % (km,
49 +     print("Fare: %d km x Rs %.2f/km x %d passengers x %.2f = Rs %.2f  [%s]"
50 +         km, train["fares"][cls], n, premium, fare, status))
51 @@ -285,7 +303,7 @@
52 -                                     "class": cls, "booking_time": datetime.now().i
53 +                                     "class": cls, "quota": quota, "booking_time":
54 @@ -367,7 +385,7 @@
55 -     print("Railway Reservation System (Session 13 build)")
56 +     print("Railway Reservation System (Session 14 build, v2)")
```

Verification

■ Write in lab record

Run on 2026-09-26 at 09:05 with a fresh data file. New tests for CR-03 first, then the Session 13 checklist re-run on 1.1 as regression.

Test id	Test	Expected	Observed	Pass
T-19-1	Availability on a 12951 run: 2A has 48 seats	General 43, Tatkal 5	2A AVAILABLE 43 Tatkal 5	Yes
T-19-2	1A has 18 seats, 3A has 128	Tatkal 2 and 13	1A ... Tatkal 2, 3A ... Tatkal 13	Yes
T-20-1	Tatkal on S001, 10 days out	Refused before passenger entry	Tatkal opens only on the day before departure. Booking refused.	Yes
T-20-2	Tatkal on a run dated tomorrow	Accepted	Booked, status CONFIRMED	Yes
T-21-1	Tatkal fare, 2A BCT to NDLS, 1 passenger	$2.90 \times 1384 \times 1 \times 1.30 = 5217.68$	Rs 5217.68	Yes
T-21-2	Seat comes from the Tatkal pool	Label 2A1-44 or later (pool is 44 to 48)	seat 2A1-44	Yes
T-21-3	Tatkal count drops, general count unchanged	Tatkal 4, General 43	2A AVAILABLE 43 Tatkal 4	Yes
R-1	General booking fare unchanged	8027.20 for 2 passengers in 2A	$\times 1.00 = \text{Rs } 8027.20$	Yes
R-2	General waitlist and promotion on cancellation	As in Session 13	Promoted in order	Yes
R-3	Refund slabs 427.50, 243.75, 0.00	Unchanged	Unchanged	Yes
R-4	Old 1.0 data file loads in 1.1	Bookings without quota key still print	Loaded and printed	Yes

Re-demonstration Transcript

Repeated menu listings removed. The schedule created is for 2026-09-27, the day after the run.

```
1 Railway Reservation System (Session 14 build, v2)
2 Choice: 1
3 Choice: 2
4 12137 Punjab Mail      CSMT > BPL > NDLS  classes: SL, 3A
5 12951 Mumbai Rajdhani  BCT > KOTA > NDLS  classes: 3A, 2A, 1A
6 Train number: 12951
7 Run date (YYYY-MM-DD): 2026-09-27
8 Schedule S003 created: 12951 on 2026-09-27.
9 Choice: 0
10 Choice: 2
11 Choice: 2
12 Schedule id: S003
13 12951 Mumbai Rajdhani on 2026-09-27
14   3A  AVAILABLE 115  Tatkal 13
15   2A  AVAILABLE 43  Tatkal 5
16   1A  AVAILABLE 16  Tatkal 2
17 Choice: 3
18 Schedule id: S001
19 From station code: BCT
20 To station code: NDLS
21 Class (3A/2A/1A): 2A
22 Quota (G=General/T=Tatkal): T
23 Tatkal opens only on the day before departure. Booking refused.
24 Choice: 3
25 Schedule id: S003
26 From station code: BCT
27 To station code: NDLS
28 Class (3A/2A/1A): 2A
29 Quota (G=General/T=Tatkal): T
30 Number of passengers (1-6): 1
31 Passenger 1 name: Priya Menon
32 Passenger 1 age: 27
33 Passenger 1 gender (M/F/O): F
34 Passenger 1 berth preference (LB/MB/UB/SL/SU): LB
35 Fare: 1384 km x Rs 2.90/km x 1 passengers x 1.30 = Rs 5217.68  [CONFIRMED]
36 Confirm booking (Y/N): Y
37 Booked. PNR 3361509606  status CONFIRMED
38 PNR 3361509606  12951 Mumbai Rajdhani  2026-09-27  BCT > NDLS  class 2A  far
39   Priya Menon  27 F  pref LB  seat 2A1-44
40 Choice: 2
41 Schedule id: S003
42 12951 Mumbai Rajdhani on 2026-09-27
43   3A  AVAILABLE 115  Tatkal 13
```

```
44      2A  AVAILABLE 43  Tatkal 4
45      1A  AVAILABLE 16  Tatkal 2
46 Choice: 0
47 Choice: 0
48 Bye.
```

Release Note

■ Write in lab record

RRS 1.1, 2026-09-26. Implements CR-03. Adds a Tatkal quota of 10% of seats per class (rounded up), bookable only on the day before the run date, at a 30% fare premium, with no waitlist. New prompt "Quota (G=General/T=Tatkal)" on the booking form and a Tatkal free count on the availability screen. Booking records gain a `quota` attribute. Data files from 1.0 load unchanged. Known behaviour: classes with fewer than 10 seats have one seat moved to Tatkal. Deferred: CR-01, CR-02, CR-04, CR-05. Rejected: CR-06.

Re-demonstration and Sign-off

■ Write in lab record

Item	Value
Version shown	RRS 1.1, <code>rrs_v2.py</code>
Shown to	Nikhil Sharma, the same reviewer as on 2026-09-20
Date and duration	2026-09-26, 20 minutes
Scenario	Transcript above: availability with Tatkall counts, Tatkall refused 10 days out, Tatkall accepted for tomorrow at premium fare, count reduced
Reviewer remarks	Accepted CR-03 as implemented. Asked that CR-02 be next. Noted the Tatkall window opens at midnight rather than 10:00, agreed to leave it for 1.1
Regression result	All 12 Session 13 checklist rows pass on 1.1
Sign-off	Reviewer: Nikhil Sharma, 2026-09-26. Developer: student, 2026-09-26. Quality member: 2026-09-26
Status of CR-03	Closed

Viva Questions

× Do not copy. Read for understanding and the viva

- **Q:** Why implement only one of six suggestions? **A:** One change per version keeps the diff reviewable and reversible; the others were deferred with reasons, not dropped.
- **Q:** What is a baseline? **A:** The frozen set of artefacts a change is measured against: here `rrs.py`, the SRS, the data dictionary and the Session 13 checklist.
- **Q:** What did the impact analysis find that the request did not mention? **A:** The data dictionary needs a `quota` attribute, the traceability matrix gains three rows, and classes under 10 seats lose a seat to Tatkall.
- **Q:** Why is `TATKAL_SHARE` a constant and not typed by the admin? **A:** The change request fixed it at 10%; making it configurable is a different change request.

- **Q:** How was regression checked? **A:** The 12 rows of the Session 13 validation checklist were re-run on `rrs_v2.py` with the same inputs plus quota G.
- **Q:** Why is `cancel_ticket` not in the diff? **A:** `promote_waitlist` calls `free_seats` with the default general pool, so the cancellation path already excludes Tatkal seats. Impact analysis showed no edit was needed.
- **Q:** Who sits on a CCB? **A:** Someone for the customer, someone for development and someone for quality; each has a reason to say no.
- **Q:** What does the release note give the reviewer that the diff does not? **A:** The behaviour change in plain words, compatibility with old data, and what is still open.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Editing `rrs.py` in place. The baseline must survive so the diff exists and the change can be reverted.
- Implementing every suggestion at once, so the second demonstration cannot say which change caused which behaviour.
- A change request without priority, effort or a named requester. The CCB has nothing to decide on.
- Impact analysis that lists only code. The SRS, data dictionary, DFD and traceability matrix all change.
- No regression run. Verifying only the new prompt misses the case where the general pool shrank.
- Sign-off by the developer alone. The problem statement says the same person must see the updated software.

Session Summary

■ Write in lab record

- Demonstration record with reviewer, date, environment and scenario
- Table of six suggested changes with priority, effort and type
- Change request form and impact analysis for CR-03
- CCB decision table with the reason for each deferral and the rejection

- Listing of `rrs_v2.py` and the diff summary against `rrs.py`
- Verification table, re-demonstration transcript, release note and sign-off table

 [Edit this page](#)

[< Previous](#)
Session 13

[Next >](#)
Session 15