Final Report
Auto Repair and Service Management System

Team 26

Natalie Kraft | nakraft
William Grochocinski | wagrocho
Ajith Kumar Vinayakamoorthy Patchaimayil | avinaya
Aditya Srivastava | asrivas7
Leo Hsiang | yhsiang

Discussion

Our application design implements AUTOR's design considerations through thorough constraint checking to ensure data quality entering the system and inclusion of advanced SQL trigger features to maintain automation of database functionality. These implemented constraints are constructed so that the database remains in BCNF.

We can ensure that the design is in 1NF as there is no attribute domain with elements that are relations. In other words, as none of our tables are nested, our design is at least in 1NF.

Our design is in at least 2NF as all of our relations have no partial dependencies. For example, earlier in our design process, we intended to have the customer id and the vehicle id (vin) as part of our attribute field for invoice. However, when looking at this design, we understood that we would not be able to be in 2NF as one of the non-prime attributes (cid) would be dependent on a proper subset of the candidate key (vin). We thus removed this duplicative field from the invoice id to ensure we would remain in at least 2NF. Similar checks were conducted on all of our functional dependencies (see Appendix 4 for the listing) to ensure that we had removed all partial dependencies from the system design.

Furthermore, our design is in 3NF as there are no transitive functional dependencies and we meet the requirements of 2NF. Transitive functional dependencies can be seen when any changes to a non-key attribute would cause changes to another non-key attribute within the table. If this occurs, the design is not in 3NF. While there are constraints placed on the tables so that values are consistent—for example in the Service_Center table mechanic_maximum_rate must be greater than mechanic_hourly_rate—there is no attribute which would support decomposing our tables into further tables.

Lastly, the dependencies have just the superkeys determining every non-prime attribute. In addition, the design has been confirmed to uphold 3NF constraints, thus, the design is consistent with BCNF. This is beneficial to ensure that our database will not end up with lots of anomalies throughout the different tables.

Aside from our database normalization scheme, it is valuable to acknowledge the automation implemented in the database through triggers and the constraints that they helped to support. The lists of triggers and constraints can be found in Appendix I and II. While the ER diagram (see Appendix III) does showcase some of the constraints. The relationships between attributes in each of these entities cannot be captured in the ER diagram, and instead we have implemented them throughout the use of constraints and triggers in the DBMS.

First off, as Oracle SQL does not include the 'assertion' statement, checks to our logic between different tables had to occur within triggers. To ensure debugging capabilities were smooth and that errors would reject invalid statements, custom errors were created and would throw a code number to support tracing back the origin of the issue. This helped identify errors with data entry into the system.

More importantly, the triggers supported the automation of the workflow by allowing deletions, updates or insertions to occur in other tables when a certain condition was met. For example, a vehicle's last performed maintenance schedule is updated when an invoice with a maintenance schedule was created, a customer would move into good or bad standing based on their payment of an invoice and a calendar appointment would be updated with an invoice id when the calendar was scheduled. These are all examples of how the triggers supported automating the database so that no checks were needed on the frontend to manage the system's database functionality.

It is also valuable to note that constraints that involved only one table were included in the original DDL for that table. These constraints included things like making sure the date and time slots were in valid ranges, ensuring the employee id numbers were the valid length and that the vehicle id was only allowed to be 8 alphanumeric characters.

The command line interface was responsible for providing some checks on the system when displaying certain tables. For example, the front end had to check to make sure none of the displayed time slots to schedule a service would end up resulting in a mechanic being overworked. However, this check is also being run on the back end before a service can be scheduled (see *invoice_checks* in Appendix II), it is simply that it also must be checked on the front end when displaying the list of valid times.

In conclusion, AUTOR has been developed with a BCNF design in mind, with autonomous triggers to support database requirements and constraints to ensure data integrity.

Appendix

I Constraint Table

Sentence	Constraint	Implementation
Each service center is identified by a globally unique ID, address, and a telephone number.	Service centers have a service center ID (sid) as their primary key. Their address and telephone numbers are unique to all service centers.	Service Center Table DDL: telephone NUMBER(10) UNIQUE address VARCHAR(100) UNIQUE PRIMARY KEY (sid)
Each center operates 5 days a week (M-F) from 8 AM to 8 PM. Some (not all) are also open on Saturdays from 9am - 1pm.	Service center's opening hours are listed in a variable 'saturday' in the Service_Center table. Valid responses are constrained to 'o', 'open', 'c', 'close'.	Service Center Table DDL: saturday VARCHAR(5) DEFAULT 'close' CHECK (saturday IN ('open', 'o', 'close', 'c'))
The calendars must maintain consistency with the opening of the service schedule.	Service centers not open have no ability: - For mechanics to schedule time off on saturday - For invoices to be created on a saturday Service centers open on Saturday must - Only allow mechanics to take time off during open hours (2, 3, 4) on Saturday - Only allow invoices to be scheduled during open hours	Implemented constraints throughout triggers: *time_slot_checks, mechanic_requests_time_off, invoice_checks*. Ensured other functionality was constrained to custom service center open logic, such as the generating the available time slots for Mechanics.
Each center has its own hourly rate for mechanics. These rates are logical and ensure a mechanic cannot have a rate outside allowable ranges.	Service Center rates are checked to be logically sound.	Service Center Table DDL: CHECK (mechanic_maximum_rate >= mechanic_minimum_rate), (if mechanic_hourly_rate is not null) CHECK (mechanic_maximum_rate >= mechanic_hourly_rate AND mechanic_minimum_rate <= mechanic_hourly_rate)

Each service center has exactly 1 manager, exactly 1 receptionist, and at least 1 mechanics. Managers, mechanics, and receptionists are all employees and can only hold one role.	Service Centers cannot have multiple managers or receptionists.	Receptionist & Manager Table DDL: sid NUMBER(10) UNIQUE In the Receptionist and Manager table, sid is unique to prevent multiple receptionists/managers being allowed to belong to the same Service Center. Trigger *add_manager_as_employee* ensures a manager is added as an employee so that exactly 1 manager exists. Trigger *employee_isa* ensures managers, mechanics, and receptionists are all a valid type of employee role.
Service centers are available if they meet a certain set of criteria.	Service Centers must have 3 mechanics, a receptionist and hours set for saturday for the field 'available' to be marked 'available'	The trigger *employee_isa* checks this constraint and updates the field when it meets the constraint.
Each employee must have a locally unique 9-digit employee ID.	Each employee has a partial key (eid), and has a primary key consisting of their partial key (eid) and the service center at which they are assigned (sid). Their eid is constrained to be 9 digits. This constraint is maintained throughout all tables with an eid.	Employee Table DDL: PRIMARY KEY (eid, sid), FOREIGN KEY (sid) REFERENCES Service_Center CHECK (eid BETWEEN (100000000 and 999999999)
Each employee is associated with only one service center.	Employees have a service center id (sid) as a foreign key and as part of the primary key (eid, sid). The employee is a weak entity of the service center.	Employee Table DDL: PRIMARY KEY (eid, sid), FOREIGN KEY (sid) REFERENCES Service_Center
There are three bundles or schedules, Schedules A, B	A schedule can be added if it's A, B, or C type. If a	Services Table DDL: serviceName VARCHAR(1)

and C.	schedule field is NULL, then the service is only a repair.	CHECK (serviceName IN ('A', 'B', 'C'))
Individual services belong to only one of the 6 repair service subcategories.	Services have a repair category attribute that can be null (denoting a maintenance service), ‘Engine Services’, ‘Exhaust Services’, ‘Electrical Services’, ‘Transmission Services’, ‘Tire Services’, or ‘Heating and A/C Services’.	Services Table DDL: repair_category VARCHAR(50) CHECK (repair_category IN (‘Engine Services’, ‘Exhaust Services’, ‘Electrical Services’, ‘Transmission Services’, ‘Tire Services’, or ‘Heating and A/C Services’)) TRIGGER *maintence_isa_schedule* takes care of isa-relationship between service and maintenance.
Individual services have a globally unique number and name.	Services have a service id (number) and service name (varchar) as their primary key.	Services Table DDL: PRIMARY KEY (serviceName, serviceNumber)
The price for each service may vary in different stores; however, the duration for each service is the same in every store. Each service also has a price and a time estimated for the service job, which is based on the car and the specific auto center.	Cost Details has a weak relationship with both Service Center and Work Event (which Services and Schedule are in a isa-relationship with), and has a manf attribute for determining pricing based on vehicle manufacturer. Duration Details has a weak relationship with Work Event and has a manf attribute for determining duration based on vehicle manufacturer. Duration is not constrained based on the service center.	Cost_Details Table DDL: manf VARCHAR(50) FOREIGN KEY (sid) REFERENCES Service_Center(sid), Duration_Details Table DDL: manf VARCHAR(50), dur INTEGER
Schedules include a set of individual services in a bundle that are completed together and are priced as a bundle.	A schedule has a one-to-many relationship with maintenance services.	The Maintenance_Schedule table has a many-to-many relationship between schedules and maintenance services.
A Customer has id (an integer) that is unique with respect to a specific center.	A customer has customer id (cid) as a partial key, and together service center ID	Customer Table DDL: PRIMARY KEY (cid, sid)

	(sid) and customer id (cid) form a primary key.	
A vehicle is identified by a globally unique id of 8 alphanumeric characters.	Vehicles have a VIN as their primary key.	Vehicle Table DDL: vin VARCHAR(10) CHECK (REGEXP_LIKE(vin, ‘^[A-Za-z0-9]{8}$’)
A customer is associated with at least one vehicle.	Each vehicle belongs to exactly one customer ID (cid) at one service center (sid).	Vehicle Table DDL: PRIMARY KEY (vin), FOREIGN KEY (cid, sid) REFERENCES Customer
Each service event (all services scheduled by a customer in a single visit to the interface) is handled by a single mechanic.	Each invoice (id) has a relationship with a single mechanic (eid) via a work relationship, but a mechanic can have a relationship with multiple invoices.	Invoice Table DDL: PRIMARY KEY (id) FOREIGN KEY (eid, sid) REFERENCES Mechanic
Invoice durations must be the same length as the sum of all services scheduled for that invoice.	Each invoice lists the time slots that it is scheduled during, these time slots are the same quantity sequentially as the length of the services.	The trigger *invoice_checks* validates that the invoice is scheduled for the correct length of time before the invoice can be created.
Invoices can only have 1 schedule listed. This schedule must be the next schedule needed for the vehicle.	Each invoice can only have one schedule included and this must be the schedule that the vehicle needs next.	The trigger *invoice_checks* validates that the invoice only has one schedule and that the invoice has the correct upcoming schedule to perform.
Invoices can only be scheduled for a mechanic who is available during that time, who is not on vacation and who is not overworked.	Mechanics need to meet all characteristics of availability before the invoice is scheduled.	The trigger *invoice_checks* validates that the mechanic is available for the full length of the service.

II Trigger Table

The following list of triggers that we utilized to implement constraints and propagate data can be described as follows.

Trigger Name	Occurs When	Description
add_manager_as_employee	After insert on Service Center	After a service center is added, we make sure each service center's manager is considered as an employee.
employee_isa	After insert on Employee	After an employee is added, we associate this employee based on his/her role to the correct table. In addition, this trigger checks if the service_center meets the criteria to be 'available'.
service_isa	Before insert on Services	Before a service is added, we ensure its categories are valid (either a maintenance service or the repair categories must be valid) and then assign this service to a work event. It is done prior to adding the schedule to make sure there are no primary key conflicts for the work events, which all have duration and cost details.
schedule_isa	Before insert on Schedule	Before a schedule is added, we assign this service to a work event. It is done prior to adding the schedule to make sure there are no primary key conflicts for the work events, which all have duration and cost details.
maintanence_isa_schedule	After insert on Services	After a maintenance service is added, we associate this service with the schedule by adding the maintenance to the Maintenance_Schedule table to ensure we have a listing of the bundle of individual maintenance services.
rate_in_check	Before update on mechanic	Before a mechanic's information is updated, we ensure the new hourly rate is valid. Each

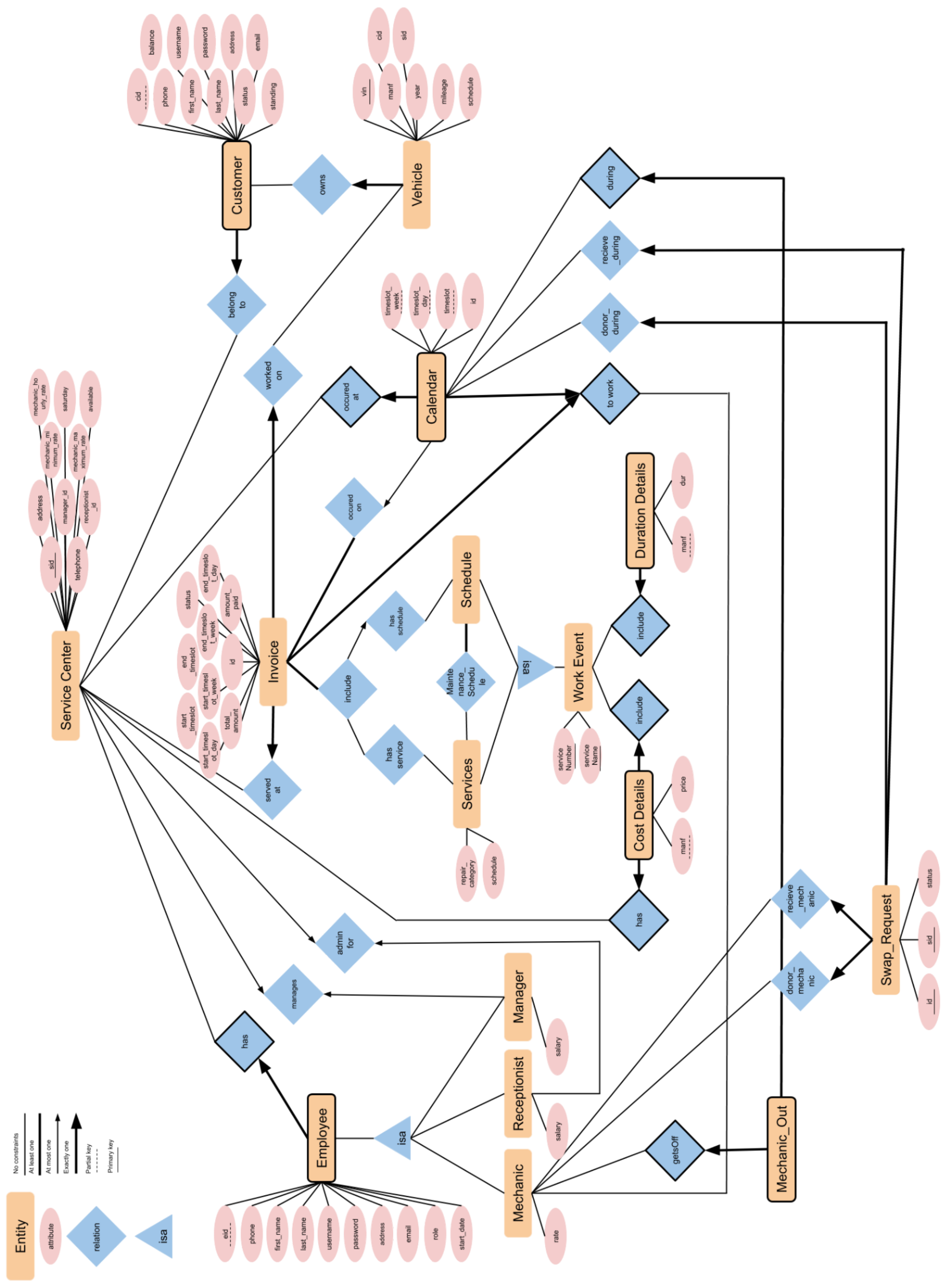
		center has its own hourly rate range set for mechanics, so this check is needed.
time_slot_checks	Before update on calendar	We must ensure the new timeslot is valid. It would be invalid if the new timeslot day is on Saturday but the associated service center does not open on Saturday. Moreover, even if the service center opens on Saturday, it would also be invalid if the new timeslot is not within 2-4(9am to 1pm).
receptionist_for_center	Before insert on employee	If the new employee role is receptionist, we associate this receptionist to the service center. There can only be one receptionist per service center.
mechanic_requests_time_off	Before insert on mechanic_out	For a mechanic's time off to be automatically approved, we need to ensure it meets all constraints. First, the new time off slot must be valid. The time slot has to be during the service center's business hours and mechanic's original working hours. Moreover, we need at least 3 mechanics working at any given time. The mechanic must also not have any invoices scheduled to occur at that time. If these conditions are not met, the time off request will be denied. If the time off is approved, the calendar will be updated by removing the available calendar slot for this employee from the calendar listing. This trigger is needed for the workflow of whether a mechanic's time off request can be granted. An assumption is made here that once a mechanic requests the time off and it is approved, they cannot cancel the request and work at that time.
generate_mechanic_schedule	After insert on mechanic	After a mechanic is added, we generate a calendar that includes all the business hours of the service center for the mechanic. This also creates a unique temporal id for the

		calendar to ensure that we have a counter to propagate invoices across weeks and days.
invoice_checks	Before insert on invoice	To ensure an invoice is accurate, it must meet the following conditions. First, the invoice must be scheduled for a valid time period (error 20001/20002 will be thrown if this is not the case). Second, the length of the invoice schedule needs to match the duration for the services to be scheduled for this invoice and the mechanic must not have requested any time off during this block of time (error 20003 will be thrown if this is not the case). Third, if a schedule is being scheduled, it must be the next schedule needed for the vehicle (error 20005 will be thrown if this is not the case). Fourth, the scheduling of this invoice should not cause the mechanic to overwork in any given week that the invoice is being scheduled during (error 20004 will be thrown if this is not the case). This trigger also updates some other fields. This includes finding the total price of the services to be performed, and setting the status of the invoice depending on if anything has been paid on the service. It also updates the balance for the customer to ensure that the customer's standing (good or poor based on their payment of invoices) is up to date.
invoice_propogate	After insert on invoice	After an invoice is added, we must update the associated calendar with the new invoice id during the appropriate time slots for the mechanic working. In addition, if the invoice includes a schedule, the vehicle's 'last schedule performed' is updated.
decide_status_invoice	After update on invoice	After an invoice is updated, update its status if the paid amount is equal to the total amount.

update_customer_balance	After update on invoice	A customer’s balance tracks the difference between the total amount their invoices have charged and the total amount they have paid. This change to the balance propagates a change in the customer’s good standing.
determine_customer_standing	After update on customer	Once the customer’s balance is updated, if the balance drops to 0, their standing becomes 1. If they schedule a new invoice, their standing will drop back to 0 if that invoice is unpaid (the latter logic is implemented in the *invoice_checks* trigger).
request_swap	Before insert on mechanic_swap_request	Before adding a new swap request, certain conditions must be passed. First, the mechanics scheduled to work must actually be scheduled to perform an invoice at the time provided for the switch (error 20001 thrown if not the case). Second, at the time requesting the switch, the other mechanic must not be scheduled to perform on another invoice (error 20002 thrown if not the case). Third, both employees must be working less than 50 hours for any week that the swap would occur during if the swap would occur. This takes into account losing time that week if the swap occurs during the same week (error 20003 thrown if this is not the case). Fourth, both mechanics must not be scheduled to be on vacation on the requested times to switch (error 20004 thrown if not the case).
update_swap	After update on mechanic_swap_request	If a mechanic swap is allowed, the mechanic asking to swap must first approve or reject the request. If the request is rejected, nothing is changed except the status on the swap request table. If they approve, then many changes are needed across the database. First, the mechanic ids on the invoices that are being switched must be switched. Secondly, the

		invoice_id on the calendar must be switched between the two mechanics during the same time period.
vehicle_makes_customer_active	After insert on vehicle	After a vehicle is added to the table, increase the number of vehicles associated with this customer. If the count is greater than 0, it is assumed the customer is active.
vehicle_makes_customer_inactive	After delete on vehicle	After a vehicle is deleted from the table, decrease the number of vehicles associated with this customer. If the count reaches 0, it is assumed the customer is inactive.

III Extended ER Diagram



IV Extended Functional Dependencies

SERVICE_CENTER

- (sid) → telephone
- (sid) → address
- (sid) → mechanic_minimum_rate
- (sid) → mechanic_maximum_rate
- (sid) → mechanic_hourly_rate
- (sid) → saturday
- (sid) → manager_id
- (sid) → receptionist_id
- (sid) → available

EMPLOYEE

- (eid, sid) → phone
- (eid, sid) → first_name
- (eid, sid) → last_name
- (eid, sid) → username
- (eid, sid) → password
- (eid, sid) → address
- (eid, sid) → email
- (eid, sid) → start_date
- (eid, sid) → role

MECHANIC

- (eid, sid) → rate

RECEPTIONIST

- (eid, sid) → salary

MANAGER

- (eid, sid) → salary

COST_DETAILS

- (manf, sid, serviceName, serviceNumber) → price

DURATION_DETAILS

- (manf, serviceName, serviceNumber) → dur

SERVICES

- (serviceName, serviceNumber) → repair_category, schedule

CUSTOMER

- (cid, sid) → phone
- (cid, sid) → first_name
- (cid, sid) → last_name
- (cid, sid) → status
- (cid, sid) → balance
- (cid, sid) → standing
- (cid, sid) → address
- (cid, sid) → email
- (cid, sid) → username
- (cid, sid) → password

VEHICLE

- (vin) → manf
- (vin) → year

(vin) → mileage
(vin) → schedule
(vin) → cid
(vin) → sid

INVOICE
(id) → total_amount
(id) → amount_paid
(id) → sid
(id) → start_timeslot_week
(id) → start_timeslot_day
(id) → start_timeslot
(id) → end_timeslot_week
(id) → end_timeslot_day
(id) → end_timeslot
(id) → date_generated
(id) → vin
(id) → eid
(id) → status

CALENDAR
(timeslot_week, timeslot_day, timeslot, sid, eid) → id
(timeslot_week, timeslot_day, timeslot, sid, eid) → invoice_id

MECHANIC_SWAP_REQUEST
(id, sid) → donor_eid
(id, sid) → recieve_eid
(id, sid) → donor_timeslot_day
(id, sid) → donor_timeslot_week
(id, sid) → donor_timeslot_begin
(id, sid) → donor_timeslot_end
(id, sid) → recieve_timeslot_day
(id, sid) → recieve_timeslot_week
(id, sid) → recieve_timslot_begin
(id, sid) → recieve_timeslot_end
(id, sid) → status

* Tables where the primary key represented the full set of attributes in the table were omitted from this listing.

V README.md

You can find the AUTOR readme.md for group 26 at the public github here (https://github.com/nakraft/AUTOR).