Analysis procedure.

1. Read requirements.
2. Think of solutions.
3. Find missing knowledge areas.
4. Revise solutions.
5. Start design.
6. Start implementation.
7. Test implementation.
8. Submit.

Insights.

1. All League has a similar structure. They are built of teams and has a tournament season. Building this application can apply to all sport leagues.
2. Statistics may differ between different sport types, so it is better to let statistics be a dynamic record containing a single statistic value, instead of structured complete statistic record.
3. Current statistics requirements does not request for detailed information such has points types (penalty, field, 3points), it also does not include attempts to get success/fail statistic percentage.
4. This application requirement is for a single tournament season. History of tournament seasons is not required but may rise in the next version.

Some key points.

1. The application receives events from different ongoing games, it may require load balancing and dynamic instances. The events can describe game status (start, end, pause…), player statistics (start/end playing, points, …). Multiple games can update the application at the same time, so games need to be uniquely identified. Players, teams have also unique identification. The application does not support team events like timeouts.
2. Application subscribes number may be unlimited, so management API should allow load balancing and dynamic allocation of management module.
3. Following #1 & #2 it is recommended to set a part modules and services for Event input and subscription APIs.
4. The current application set a REST API for subscriptions. Real application should implement FullStack providing convenient GUI.
5. Although statistic calculation and retrieval from database is a time and CPU consumption, the application does not use cached information.
6. I use simple java conventions and avoid using Spring for readability.
7. Code file structure based on functionality. There can be 2 options to set code file structure, Object or Functionality based. In Object base, all functionality related to the same data object is located at the same directory (Data, Management, Database…). In functionality base, Objects functions are located at the same directory (directory for model, database access, management…)

Database (<database.drawio>)

1. League record identifies and describes a given league it contains league name, description, unique identifier.
2. Team record identifies a single team within the League. It contains team name, description, unique identifier, and league identifier.
3. Game record describes a single match between 2 teams. It contains name, description, unique identifier, location, date, start and end time, home and visitor team identifiers.
4. Player record identifies a single player information. It contains player name, description, internal identifier, team identifier and unique identifier.
5. Statistic type records list available statistic type. It contains a name, description and unique identifier and constraints (optional) like range (Fouls 0-6, Play Time 0.0-48.0) and event values (Play Time event “Start”/”End”).
6. Statistic record describes a single statistic instance. The record contains statistic type identifier, integer value, float value, status, team identifier, player identifier and game identifier.
7. Statistic type association table for league. Map the available statistic type available for each league

Processes

1. Event input – at an ingoing game, a stream of events delivered to the application updates game statistics. Event may be on a specific player, such as points, rebounds, assists… or on the whole game (start/end). Each event contains game identifier, team identifier, player identifier, statistic type identifier, event time, and value.   
   Event input is received by Event input module. It parses the event record and passes it to Event process method.
2. Event process – method event record. It finds the event type, checks record legality (event value meet constraints), process the event and store the event statistic. Some events have special processing. Events like game start/end have impact on the ability to modify player statistics. Event process has a pool of thread processing the events.
3. Subscription API – application subscribers information request has it own management module. Each information request is processed independently. A request is identified and submitted to a handler that processes the request and produces an answer. The handler checks request information integrity and then access the database for information. The request has several sanity checks like identity checks (existence of league, team, player, game….), relational checks (team belongs to league, player belongs to team, team played in game …)

APIs (examples in <README.md>)

1. Event input - /event POST – delivers a single event in a single game at a league. POST record contains, league identifier, game identifier, player identifier(Optional – only for player statistics), time, statistic type identifier and value.
2. Player game statistic - /league/{leagueName}/game/{gameName}/player/{playerName} GET – retrieve player real time/history game statistics.
3. Player season statistic - /league/{leagueName}/player/{playerName} GET – retrieve player season statistics.
4. Team season statistic - /league/{leagueName}/team/{teamName} GET – retrieve team season statistics.

Generic conventions

1. Code written in JAVA.
2. Each method starts and ends with log information. Public methods log in info level. Private and Protected method log in debug level.
3. Project is developed in Eclipse environment.
4. Application running in AWS cloud using AWS services.

Aws services and implementation

1. Use AWS services such as
   1. RDS from PostgreSql database (<rds-postgres.yaml>),
   2. CloudWatch for logging (<cloudwatch-service.yaml>),
   3. ECS for load balancing (<ecs-league-app.yaml>),
   4. VPC for networking load balancing (<vpc-network.yaml>).