Application Information Document

for

Basketball League Management System (BLMS)

Version 1.0

Contents

1 Introduction

This document provides information regarding Basketball League Management System (BLMS).

This system stores information about league, players, teams & player contracts.

BLMS enables user to process trading among the players of different teams. Trading history information is also stored.

Here is overview of business rules of BLMS:

- Each league can have multiple seasons.
- Each league contains N teams playing in it.
- Each team can have 15 players on the team. However, it is possible that a league can decide each season what is the maximum number of players.
- Each player can play on one or multiple positions: point guard, shooting guard, small forward, power forward and a center.
- Each player has a contract with a team for a specific season. It is possible to have a one-year contract or a multi-year contract.
- Each team can spend a limited budget for their team. It means if a salary budget is $50.000.000, overall contract value should be below that value. Otherwise, a league will detect the limit was breached and a team will have to pay for a luxury tax, which is 100% of a value above the limit. League checks contract budgets.
- Teams can trade players. It is important to know when teams are doing trades, a sum of player's contracts on each side must be similar. There can be a difference of 20% of overall traded value.
- Players can get injured during a season. In that case, their contract is not calculated in a budget. Also, in that case, an empty spot is available on a team roster.

Here is overview of functional requirements that BLMS must support:

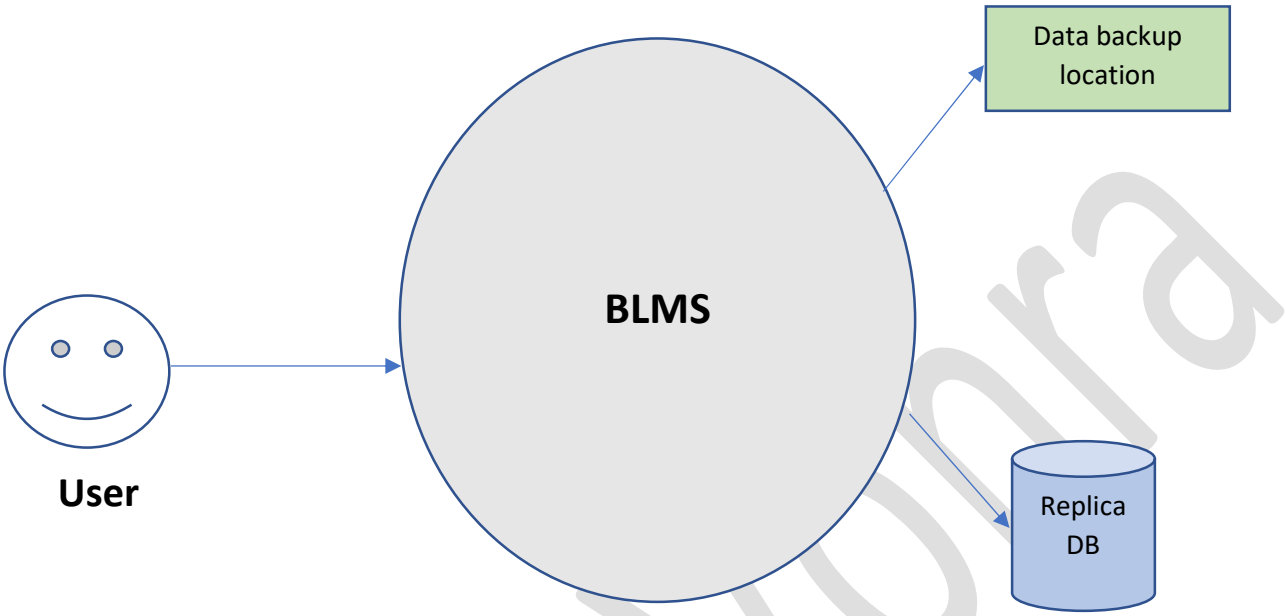
Create a basketball league management system. The database model should support storing information about the league, teams, player and their contracts and should provide reporting functionality.

1. SQL script that will create the whole database along with the constraints and relationships. Also create a function which generates sample data.
2. Create a function which places a player on an injury list. Also, create a function or the same one to remove a player from an injury list.
3. Create a function or a procedure to create trade between two teams. Allow trading multiple players from each side.
4. Create a function which will provide information about the most expensive starting lineup for a specific team. A starting lineup has one player on each position and it has to return five players, one for each position.
5. Create a function which provides monthly validation if some of the teams breached a contract limit. This function should generate luxury tax record.
6. Create a query which provides information which teams went over the budget limit for during the season.
7. Create a list of most expensive teams and most expensive player.
8. Read - replica database on the same machine. Replication needs to be as real-time as possible.

9. Automatic backups of the database every two hours.

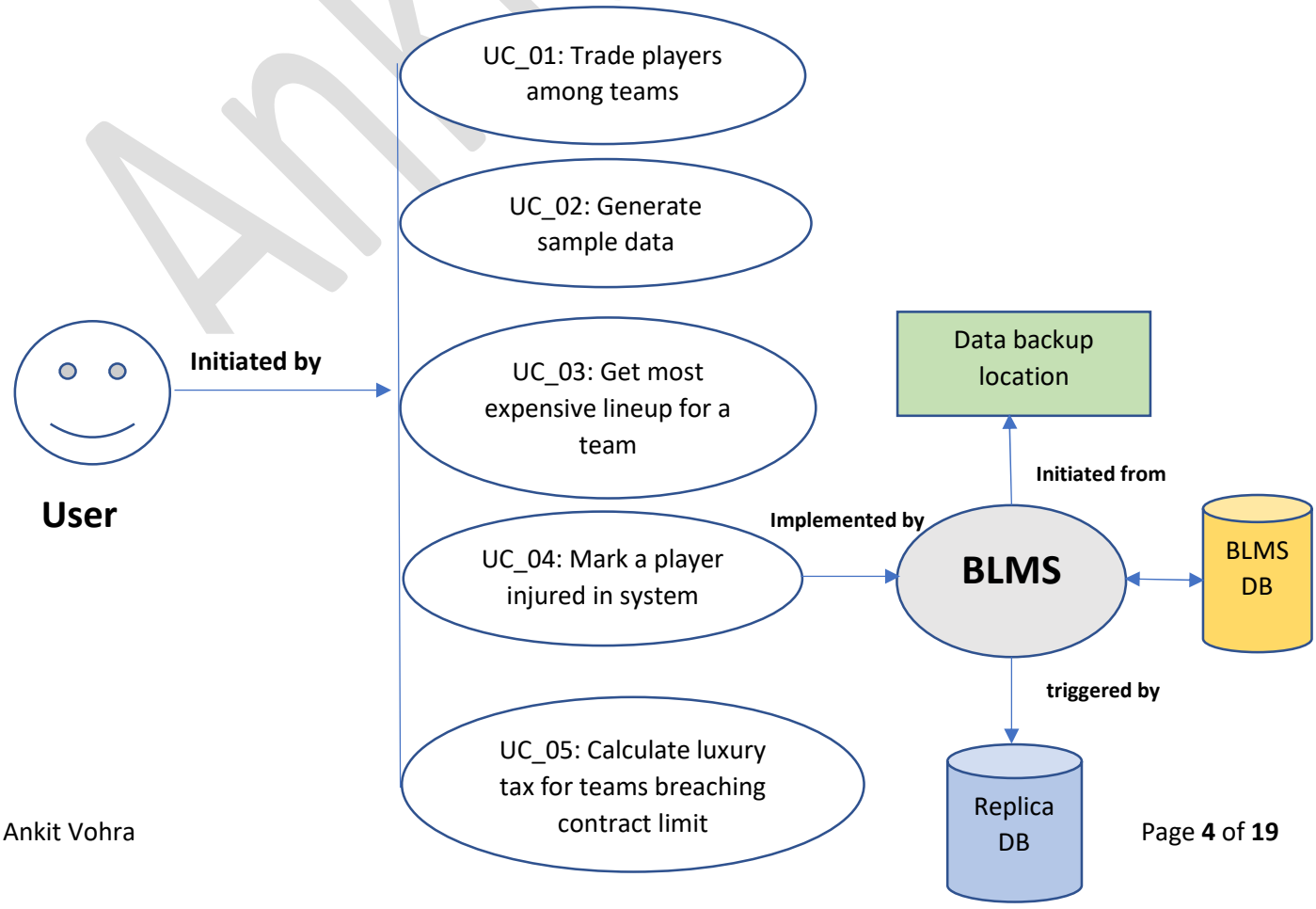
2 System Context Diagram

Below diagram is System Context for BLMS showing it as black box and the external actors that interact with BLMS.



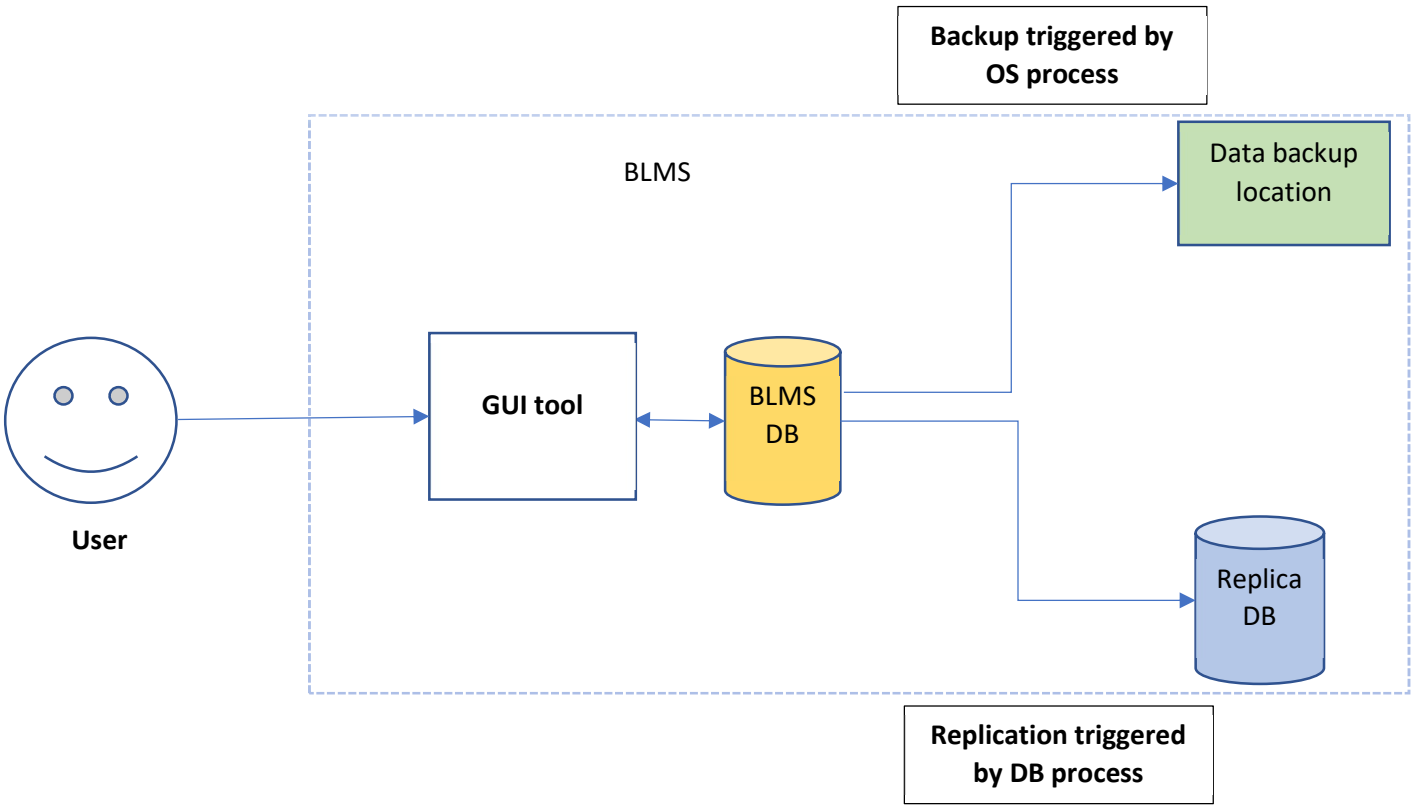
3 Use Case Diagram

Use case diagram provides more details about nature of interaction between actors & BLMS.



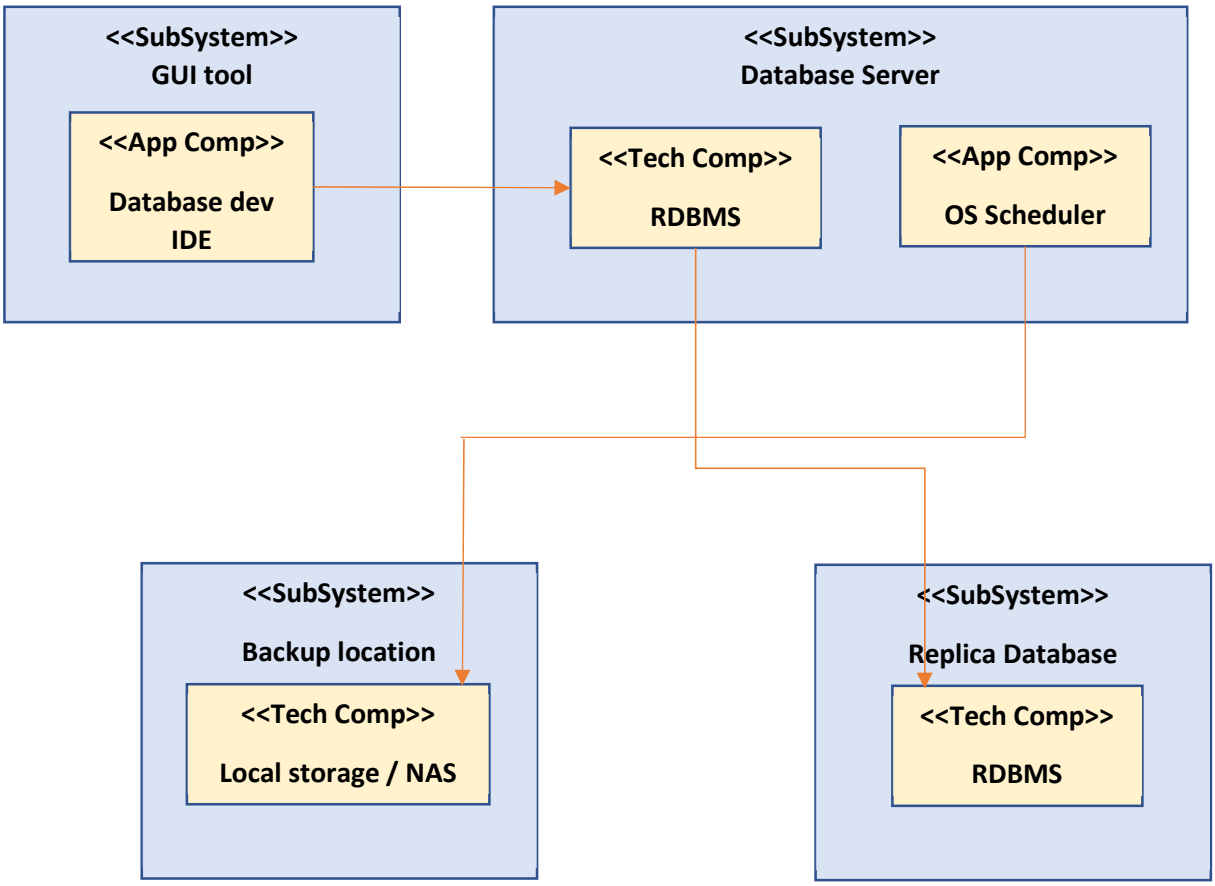
4 Architecture Overview Diagram

Architecture Overview contains the solution architecture that facilitates the understanding of solution elements and their mutual relationships.

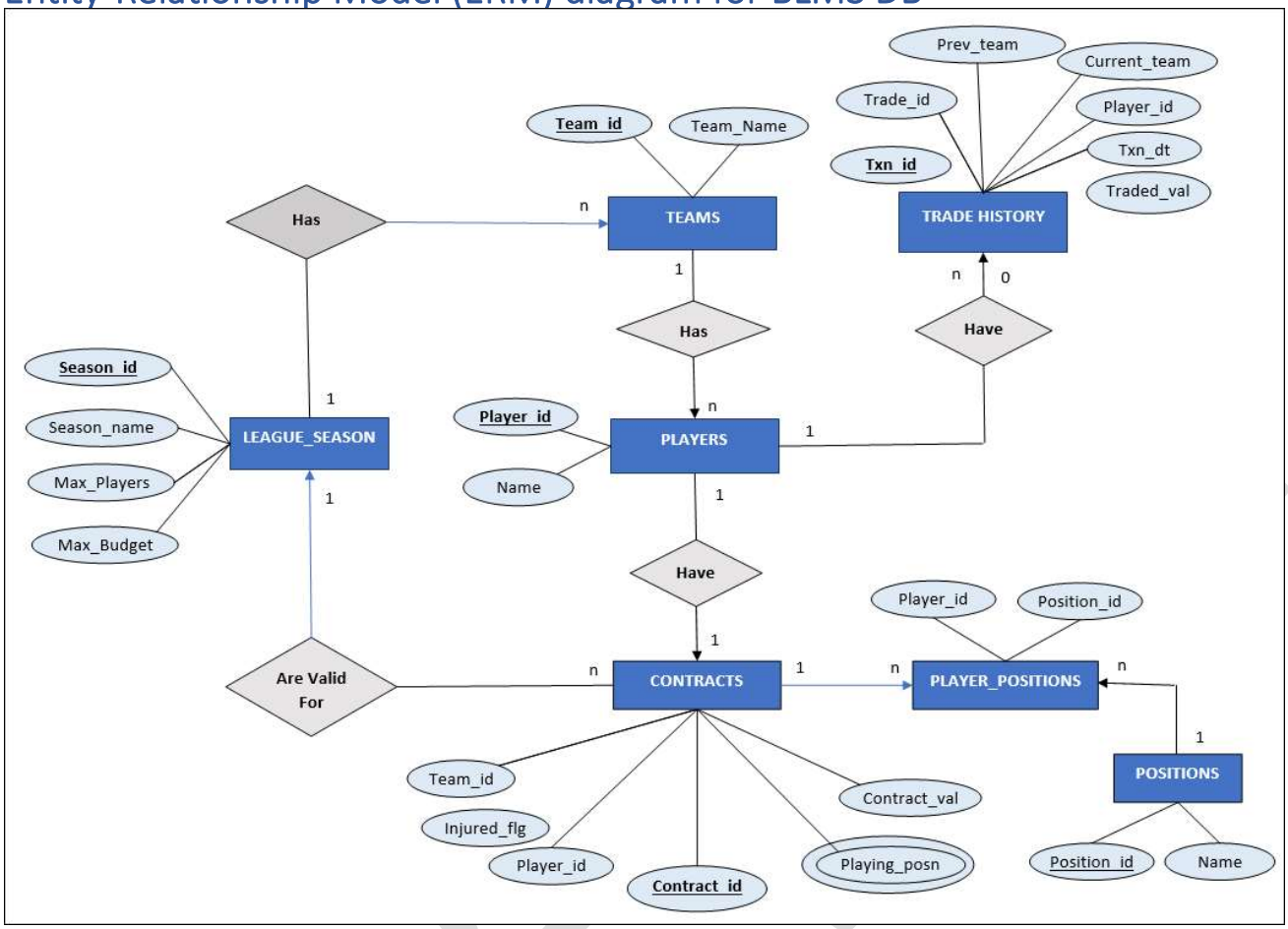


5 Functional Viewpoint: Component Model (Static View)

The static functional view describes the software components, their responsibilities, relationships & the way they collaborate to implement the required functionality.

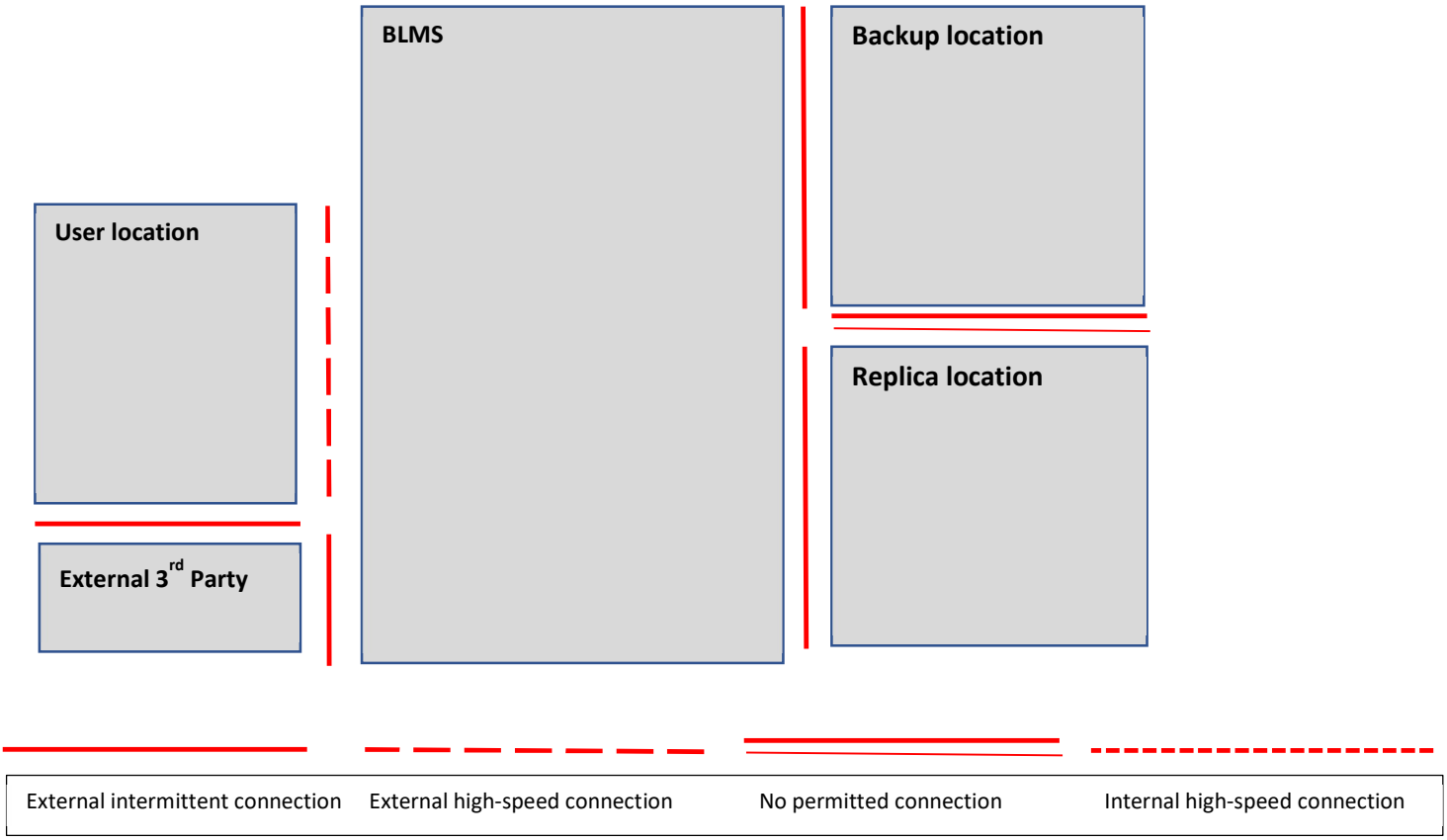


6 Entity-Relationship Model (ERM) diagram for BLMS DB



7 Operational Viewpoint: Logical Location View Model / Deployment Unit Model / Logical Operational Model

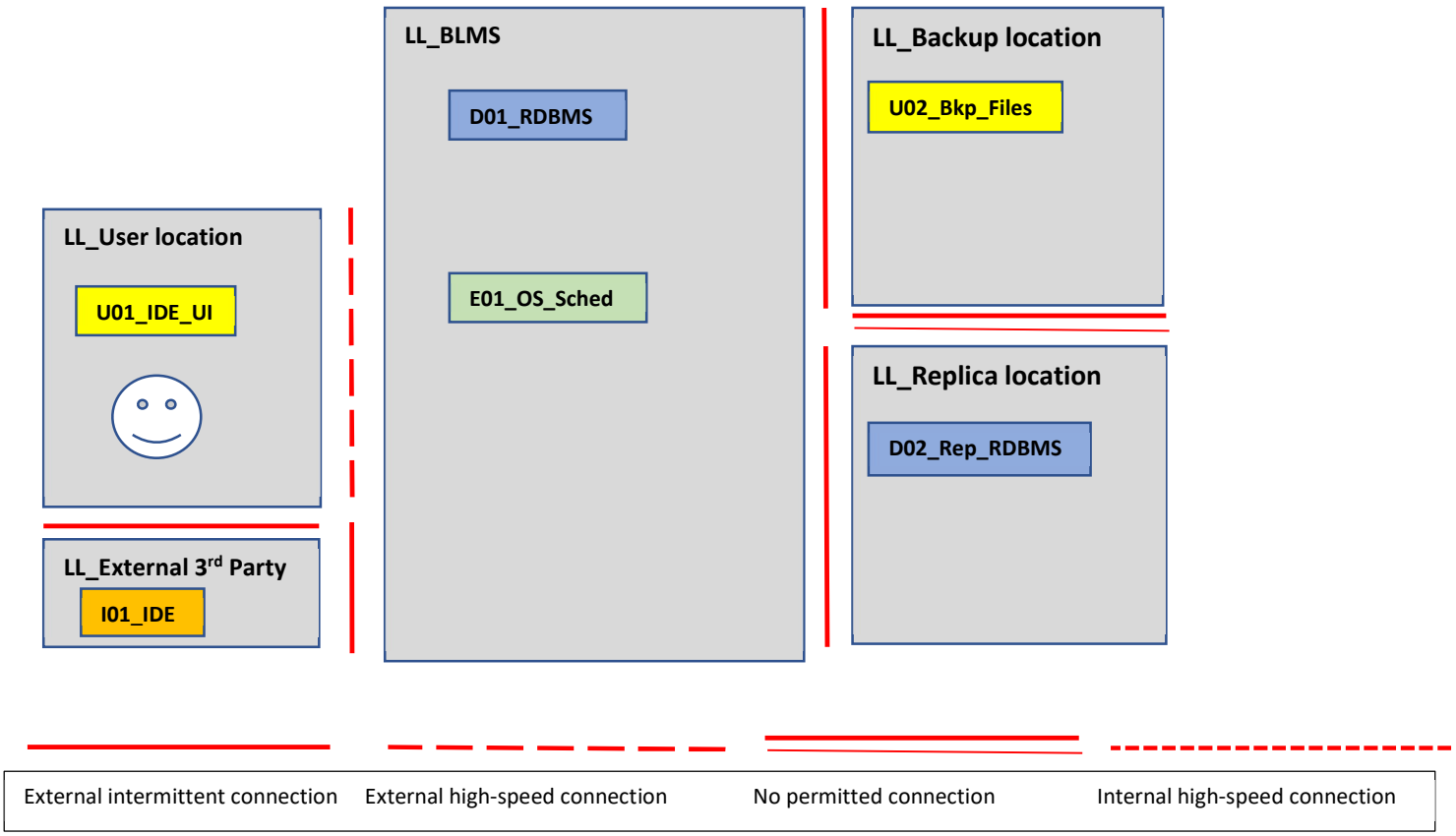
i) **Logical Location View Model**: First step in developing operational viewpoint it to develop logical viewpoint model.



ii) **Deployment Unit Model**: Below table shows DU model, which contains mapping between components (from **Component Model**) to defined **Deployment Units** (Presentation, Data, Execution OR installation)

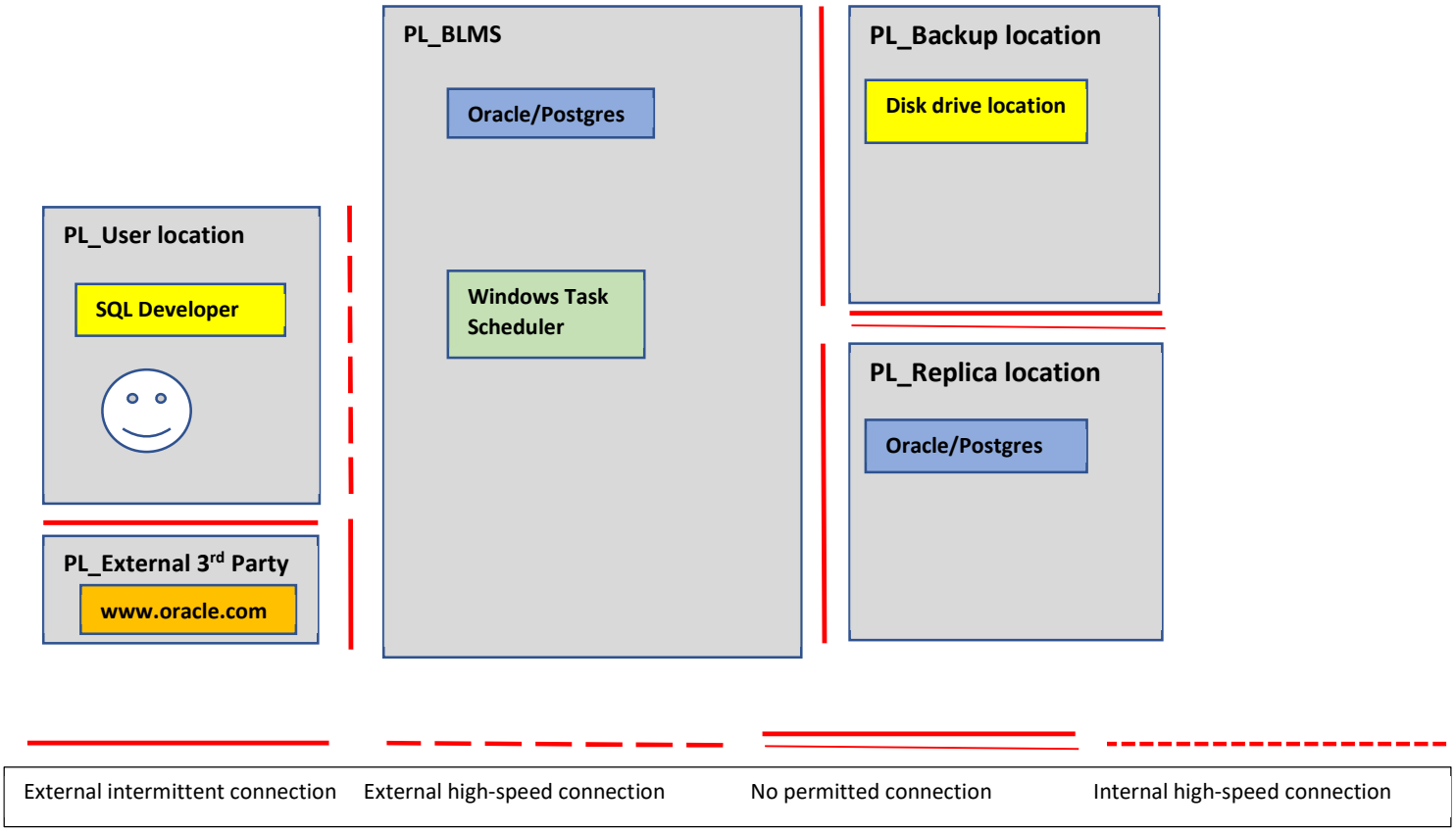
Sub-system/Layer	Component	Presentation_DU	Data_DU	Execution_DU	Installation_DU
GUI tool	Database development IDE	U01_IDE_UI			I01_IDE
Database Server	RDBMS		D01_RDBMS		
	OS Scheduler			E01_OS_Sched	
Backup location	Storage	U02_Bkp_Files			
Replica Database	RDBMS		D02_Rep_RDBMS		

ii) **Logical Operational Model**: Here identified actors, along-with Logical nodes are placed into **Logical Location View Model** developed earlier. In addition, identified Deployment Units are "deployed" onto the nodes in the model.



8 Physical Operational Model

Here is physical operational view for BLMS.



9 Description of the entities and their attributes

S. NO.	ENTITY/TABLE NAME	ATTRIBUTE NAME	Description
1	**LEAGUE_SEASON_T (To store league's information)**	SEASON_ID	Sequential primary key for the entity (generated)
		SEASON_NAME	Season name
		ACTIVE_SEASON	Flag to indicate if the season is Active (Y or N)
		MAX_PALYERS_NUM	Max number of players for the season
		MAX_SEASON_BUDGET	Max budget authorised for the season
2	**TEAMS_T (To store data for teams)**	TEAM_ID	Sequential primary key for the entity
		TEAM_NAME	Team name (generated)
3	**PLAYERS_T (To store data for players)**	PLAYER_ID	Sequential primary key for the entity (generated)
		NAME	Player name (generated)
4	**CONTRACTS_T (To store Contracts data for season)**	CONTRACT_ID	Sequential primary key for the entity (generated)
		PLAYER_ID	Player's ID (FK --> PLAYERS_T.PLAYER_ID)
		TEAM_ID	Team's ID (FK --> TEAMS_T.TEAM_ID)
		SEASON_ID	Team's ID (FK --> LEAGUE_SEASON_T.SEASON_ID)
		PLAYING_POSITION	Stores valid playing position for the player. This is a multi-valued attribute. Allowed values: 1. 'Point Guard', 2. 'Shooting Guard', 3. 'Small Forward', 4. 'Power Forward', 5. 'Center'
		INJURED_FLAG	Flag to indicate if the player is Injured (Y or N)
		ANNUAL_CONTRACT_VALUE	Annual contract value for player for given season
		CONTRACT_START_DT	Contract start date
		CONTRACT_DUR_YR	Contract duration in years
5	**TRADE_HIST_T (To store player trade history)**	TXN_ID	Sequential primary key for the entity (generated)
		TRADE_ID	Consistent id for 'N' transactions in a trade
		PLAYER_ID	Player's ID
		CURRENT_TEAM	Current team of traded player
		PREVIOUS_TEAM	Previous team of traded player
		TRADED_VALUE	Value at which given player was traded in txn
		TXN_DT	Date when trade happened
6	**PLAYER_POSITIONS_T** (players & positions. Resolves n-n relationship b/w Contracts & Positions as 1 player can have n positions & 1 position may be associated with n players)	PLAYER_ID	Player ID
		POSITION_ID	Position ID
7	**POSITIONS_T (master table to store positions)**	POSITION_ID	Position ID
		DESCRIPTION	Position name

10 Functionalities supported & technical details

1. SQL script that will create the whole database along with the constraints and relationships.

A master script ***master.sql*** will create required DB objects for implementing BLMS database.

Usage: @master.sql

2. Create a function which generates sample data

Function ***fn_generate_data*** has been created to generate sample data for the database tables.

Input Parameters: Number of teams in league, Number of max players & max budget allowed for a season

Usage: select fn_generate_data (8 , 15 , 70000000) from dual;

Return value : This returns VARCHAR2 string mentioning the status of operation (success or failure message).

3. Function which places a player on an injury list. Also, to remove a player from an injury list.

Function ***fn_toggle_injured*** has been created for this.

Input Parameters: It takes player id as input, marks player injured & returns success/failure message.

Usage: select fn_toggle_injured(<player_id>) from dual;

4. Function or a procedure to create trade between two teams. Allow trading multiple players from each side.

Function ***fn_trade_pl*** has been created for enabling trade of multiple players among two teams & maintaining trading history.

It performs a number of business validations before successful trading:

1) Check validity of team ids
2) Check if teams are full
3) Check if entered players belong to same team , are valid players & are not injured
4) Check if contract values are within permissible limits

Input Parameters: 1st team id, 1st team's player id's as string, 2nd team id, 2nd team's player ids as string.

Usage: select * from fn_trade_pl (1,'102,103', 8,'206,207');

Function returns a table type object with appropriate failure reason or success message with trade value.

5. Function which will provide information about the most expensive starting lineup for a specific team. A starting line-up has one player on each position and it has to return five players, one for each position.

Function ***fn_ret_expensive_lineup*** has been created for this.

Input Parameters: It takes team id as input and returns most expensive line-up for that team in table format. It takes into account that 1 player may play at multiple positions.

So in case of a player with high budget who can play in multiple positions, function will return record for only 1 position for that player.

Usage: select * from table (fn_ret_expensive_lineup(5));

6. Function which provides monthly validation if some of the teams breached a contract limit. This function should generate luxury tax record.

Function ***fn_lux_tax*** has been created and it report teams with exceeding budget limit & calculates luxury tax.

Usage: select * from table (fn_lux_tax);

It returns returns a table type object with luxury tax data for teams who have breached budget limit.

A procedure ***proc_lux_tax*** has been created which has been scheduled via a DB scheduler job (GENERATE_LUXURY_TAX_RECS) to run on a monthly basis at 1 AM on 1st of every month.

This procedure will generate a file with name format ***LUXURY_TAX_MMDDYYYY_HHMISS.txt*** in a directory with luxury tax data for teams who have breached budget limit.

7. Query which provides information which teams went over the budget limit for during the season.

The SELECT query will sum-up contract values for non-injured players for each team and report if the value exceeds allowed budget limit for the season.

8. A list of most expensive teams and most expensive player.

The SELECT query will sum-up contract values for non-injured players for each team and return the most expensive team. If multiple teams have same max sum value, all will be returned.
Similarly, record(s) for most-expensive non-injured player will be returned. If multiple players have same max sum value, all will be returned.

9. Read - replica database on the same machine. Replication needs to be as real-time as possible.

A replica database has been setup for the purpose.

Real time replication has been set-up through "DB link - trigger" mechanism.

10. Automatic backups of the database every two hours.

Windows Task Scheduler Job ***BIHOURLY_DB_BACKUP*** will run script "Bi-hourly_backup_orcl.bat" every 2 hrs and takes logical backup of DB schema in a directory.

11 Test Cases

S. No.	Description	Action	Expected Result	Actual Result
1	**Main DB**: Ensure a running Oracle DB instance with schema BLMS	Check DB & schema's existence & login to schema	BLMS schema exists & login is successful	do
2	**Replica DB**: Ensure a running Oracle DB instance with schema BLMS_REP	Check DB & schema's existence & login to schema	BLMS_REP schema exists & login is successful	do
3	Login to main DB with BLMS id	Login to BLMS. All below steps will have to be executed in BLMS schema unless mentioned.	Login successful	do
4	In Main DB's BLMS schema, execute script to create required DB objects & verify that all objects have VALID status.	Execute **master.sql**	Script executed successfully & all created objects in BLMS schema are in VALID state	do
5	In Replica DB's BLMS_REP schema, execute script to create required DB objects & verify that all objects have VALID status.	Execute **master_rep.sql**	Script executed successfully & all created objects in BLMS_REP schema are in VALID state	do
6	Generate test data for following input parameters: No. of teams = 10 No. of max players per team = 15 Max budget for season = 70 Mn	Run: select fn_generate_data (10 , 15 , 70000000) from dual;	Message "Data generated successfully." Will be returned. **LEAGUE_SEASON_T** table will have few rows with ONLY 1 row with column values as: ACTIVE_SEASON='Y' MAX_PLAYERS_NUM=15 MAX_SEASON_BUDGET= 70000000 **TEAMS_T** table will have 15 rows **PLAYERS_T** table will have 150 rows **CONTRACTS_T** table will have 150 rows (i.e. 1 contract row for each player) for active season id. Each player will be assigned to a team with a PLAYING_POSITION. INJURED_FLAG will be N for all ANNUAL_CONTRACT_VALUE for each player will be set as random value b/w 1 Mn – 9 Mn (just a random value rounded to nearest Mn)	do

7	Verify that generated data is replicated in **Replica** DB.	Verify same data existence in Replica DB schema by executing following in BLMS_REP schema: select * from LEAGUE_SEASON_T; select * from TEAMS_T; select * from PLAYERS_T; select * from CONTRACTS_T;	Data will be same as in Main DB	do
8	Mark a player as injured	Execute: select fn_toggle_injured(128) from dual;	Message “Success: Marked Injured” will be returned and following query will return count as 1 indicating player was successfully marked injured: select count(1) from CONTRACTS_T where player_id=128 and injured_flag='Y' ;	do
9	Verify that updated data is replicated in Replica DB.	In BLMS_REP, run following query: select count(1) from CONTRACTS_T where player_id=128 and injured_flag='Y' ;	Count value 1 will be returned	do
10	Trade players by passing in INVALID 1st team id, say id 11	Run: select * from fn_trade_pl (11,'102,103', 8,'206,207') ;	Message “Failed : Invalid team 11” will be returned as team 11 is invalid. We have team ids upto 10 in TEAMS_T table.	do
11	Trade players by passing in VALID 1st team id, and INVALID 2nd team id, say id 14	Run: select * from fn_trade_pl (9,'102,103', 14,'206,207') ;	Message “Failed : Invalid team 14” will be returned as team 11 is invalid. We have team ids upto 10 in TEAMS_T table.	do
12	Trade players by passing VALID team ids, say 3 & 7. Pass invalid players for team 3	Run: select * from fn_trade_pl (3 ,'102,232, 7,'206,207') ;	Message “Failed : Invalid/Injured player ID(s) for team 3” will be returned as players 102 does not exist in team 3	do
13	Trade players by passing VALID team ids, say 3 & 7. Pass VALID players for team 3 & INVALID player(s) for team 7	Run: select * from fn_trade_pl (3 ,'133,134', 7,'206, 193') ;	Message “Failed : Invalid/Injured player ID(s) for team 7” will be returned as players 206 does not exist in team 7	do
14	Pass VALID team ids and player ids. Make sure that total contract value is same for both ids from a team Make sure that that No player is injured from any team	Run: select * from fn_trade_pl (3 ,'133,134', 7,'193, 194') ;	Message “Trade Successfull. Contract values: 4000000 & 4000000” will appear citing successful trade & trade values for both teams	do
15	Pass VALID team ids and player ids. Pass 3 players for 1st and 2 for 2nd team	Run: select * from fn_trade_pl (3 ,'133,134,136', 7,'193, 194') ;	Message “Failed : team 7 does not have enough empty slots.”	do

			Will appear as team 7 does not have empty slots are full as per max_players parameter passed during data generation.	
16	Pass VALID team ids but **injured** player ids. Pass 3 players for 1st and 2 for 2nd team AND mark 1 player from 2nd team as Injured.	Run: select fn_toggle_injured(**194**) from dual; select * from fn_trade_pl (3 ,'133,134,136', 7,'193, **194'**) ;	Message "Failed : Invalid/Injured player ID(s) for team 7" Will appear as Injured player id was passed for trading.	do
17	Mark player marked in above step as NON-Injured.	Run select fn_toggle_injured(194) from dual;	Message "Success: Marked NOT Injured" will be returned and player is eligible for trading now	do
18	Pass VALID team ids and player ids. Mark 1 player from 2nd team as injured. Pass 3 players for 1st and 2 for 2nd team. All players should be VALID & Non injured. Make sure that contract value totals are unequal and there is more than 20% difference from smaller value.	Run: select fn_toggle_injured(**194**) from dual; Now attempt trading with VALID player ids: select * from fn_trade_pl (3 ,'133,134,136', 7,'193, **199**) ;	Message "Failed : Trade value mismatch: 7000000 & 12000000" will appear as trade value is not within permissible limits as per business requirement.	do
19	Pass VALID team ids and player ids. Pass 3 players for 1st and 2 for 2nd team. All players should be VALID & Non injured. Make sure that difference of contract value lies between 20% of value of lower total contract value.	Run: select * from fn_trade_pl (3 ,'133,134,136', 7,'193, **195'**) ;	Message : "Trade Successfull. Contract values: 7000000 & 8000000" will appear as 20%70000000 = 14,00,000 8000000 – 7000000 = 1000000 < 14,00,000 CONTRACTS_T table will get updated with players having updated team ids	do
20	Verify trade history	Run: select * from trade_hist_t where player_id in (133,134,136,193, 195) order by trade_id,txn_id;	Trading history, with above trade and 5 transactions (1 for each player traded), along-with current team, previous team and traded value will be present.	do
21	Verify that updated data is replicated in Replica DB.	In BLMS_REP in Replica DB, run following queries: Select * from CONTRACTS_T where player_id in (133,134,136,193, 195); select * from trade_hist_t where player_id in (133,134,136,193, 195) order by trade_id,txn_id;	Updated team_ids will be present in CONTRACTS_T. Same trading history will be available in TRADE_HIST_T in Replica DB.	do

22	Retrieve most expensive starting line-up for specific team.	Run following query for team 5: select * from table (fn_ret_expensive_lineup(5));	This will return total 5 rows, one row for each playing position with player id and those player ids will be most expensive in that team for given playing position. Injured players will be ignored. Same query should return same result in **Replica** DB.	do
23	Monthly validation if some team(s) breached contract/budget limit. This should generate luxury tax record.	Run: select * from table (fn_lux_tax); Monthly **scheduled** procedure PROC_LUX_TAX can be run via following: BEGIN DBMS_SCHEDULER.run_job (job_name =>'GENERATE_LUXURY_TAX_RECS'); END;	This will return rows with team name & Tax amount for the teams whose total contract value went above the budget limit passed during data generation. Injured players will be ignored while calculating contract value. The scheduler job will generate a file containing (name format LUXURY_TAX_MMDDYYYY_HHMISS.txt) same data as above. Same query should return same result in **Replica** DB.	do
24	Query to check which teams went over budget limit for the season	Run: select t.team_name ||' : '|| to_char(e.s) || ' M' over_budget_teams FROM teams_t t ,(SELECT team_id,sum (annual_contract_value) s FROM contracts_t WHERE INJURED_FLAG ='N' group by team_id) e WHERE t.team_id = e.team_id AND e.s> (SELECT max_season_budget FROM LEAGUE_SEASON_T WHERE active_season = 'Y') ORDER BY t.team_id;	This will return rows with team name & contract value for the teams whose total contract value went above the budget limit passed during data generation. Injured players will be ignored while calculating contract value. Same query should return same result in **Replica** DB.	do
25	Verify that teams whose budget went below budget limit are not selected in above 2 steps	Pick 2 teams from list of teams that appeared in above 2 steps. Mark few players from each team as injured so that total team budget goes below max budget limit. Mark them injured via : select fn_toggle_injured(< player_id >) from dual; Run above 2 steps again.	Message “Success: Marked NOT Injured” will appear and Player will be marked non-injured & injured_flag will get updated as ‘N’ in CONTRACTS_T table. The 2 teams will NOT appear in the result set.	do
26	Verify that updated data is replicated in Replica DB.	In BLMS_REP, run following query: select * from CONTRACTS_T where injured_flag='Y' ;	Players marked injured in above steps will appear in query result.	do
27	Get list of most expensive teams & most expensive player(s)	Run: SELECT '===== Most Expensive Players and Teams =====' FROM DUAL UNION ALL select team_name ||' : '|| to_char(e.s) || ' M' from teams_t t	The SELECT query will sum-up contract values for non-injured players for each team and return the most expensive team. If multiple teams have same max sum value, all will be returned. Similarly, record(s) for most-expensive non-injured player will be returned. If	do

		,(SELECT team_id, sum (annual_contract_value) s,row_number() OVER(ORDER BY sum (annual_contract_value) desc) r FROM contracts_t WHERE INJURED_FLAG='N' group by team_id)e WHERE t.team_id = e.team_id AND e.r=1 UNION ALL SELECT p.name ||' : '|| to_char(annual_contract_value) || ' M' FROM contracts_t c , players_t p WHERE annual_contract_value in (SELECT MAX(annual_contract_value) FROM contracts_t WHERE INJURED_FLAG='N') AND INJURED_FLAG='N' AND p.player_id=c.player_id ORDER BY 1;	multiple players have same max sum value, all will be returned. Same query should return same result in **Replica** DB.	
28	Automatic backups every 2 hours	Verify the scheduled job is present in operating system to take backup of main DB/BLMS schema	DMP files should be getting generated on a bi-hourly basis in operating system directory.	Do