Rich Queries

Fabric ledger

- **Transaction Log**: Stores the details of the transaction (the transaction logs/history) in a tamper-proof and sequential manner.

- **World state**: Stores the current state of the business object (asset).

  ❖ **LevelDB** stores chaincode data as simple key-value pairs.

  ❖ **CouchDB** is an optional, alternate state database that allows you to model data on the ledger as **JSON** and issue queries against data values rather than the keys. It allows **indexing** by which we can query large datasets.

Rich Queries

➢ **GetQueryResult**

➢ **GetStateByRange**

➢ **GetHistoryForKey**

➢ **GetQueryResultWithPagination**

Rich Queries

It allows to fetch data more easily from the ledger

GetQueryResult API

we can perform JSON queries against the data in the state database by using the **GetQueryResult** API and passing a **CouchDB query string**

CouchDB query string
selector (*json*) – JSON object describing criteria used to select documents.
sort (*json array*)– JSON array with list of field name and direction pairs. Used to sort the result based on different fields.

Indexing

❖ It is used to optimize the performance of database by minimizing number of disk accesses required when a query is processed.

❖ Indexes allow a database to be queried without having to examine every row with every query.

❖ Normally, indexes are created for frequently occurring queries allowing the data to be queried more efficiently.

❖ If sorting is required in a query, CouchDB requires an index that includes the sorted fields.

To define an index, we need to provide three points:

- **fields**: these are the fields to query

- **name**: name of the index

- **type**: always "json" in the context

Sample:File name:**indexColor.json**

```
{META-INF/statedb/couchdb/indexes}


{  "index": {"fields": ["color"]  }, "ddoc":
"indexColorDoc",  "name": "indexColor",  "type": "json"}
```

GetStateByRange

❖ Returns a range of data over a set of keys in the ledger

❖ The iterator can be used to iterate over all keys between the start key(inclusive) and end key (exclusive)

getStateByRange(start key, end key)

GetStateByRange

- getStateByRange(CAR-001, CAR-004)

- getStateByRange(CAR-005, " ")

- getStateByRange(" ", " ")

KEY	VALUE
CAR-001	{values: ….}
CAR-002	{values: ….}
CAR-003	{values: ….}
CAR-004	{values: ….}
CAR-005	{values: ….}
CAR-006	{values: ….}
CAR-007	{values: ….}

1 { CAR-001, CAR-002, CAR-003 }

2 { CAR-005, CAR-006, CAR-007 }

3 { all }

Queries on Private Data

getPrivateDataQueryResult ()

getPrivateDataByRange()

GetHistoryForKey

❖ Retrieves the history for a state

❖ Returns the set of stored values, including the transaction identifiers that performed the state update

❖ For each historic key update, the historic value and associated transaction id and time stamp are returned

GetHistoryForKey(Key)

GetQueryResultWithPagination

➢ It helps to list the records as different sets

Input Arguments:

    ➢ Query: CouchDB query string

    ➢ Pagesize: number of records to be displayed on a single query

    ➢ Bookmark: starting point for next set of records

Output:

    ➢ Iterator: query result set

    ➢ Metadata: fetched Records Count & bookmark

GetQueryResultWithPagination (queryString,pageSize,bookmark)

Automobile usecase - Chaincode
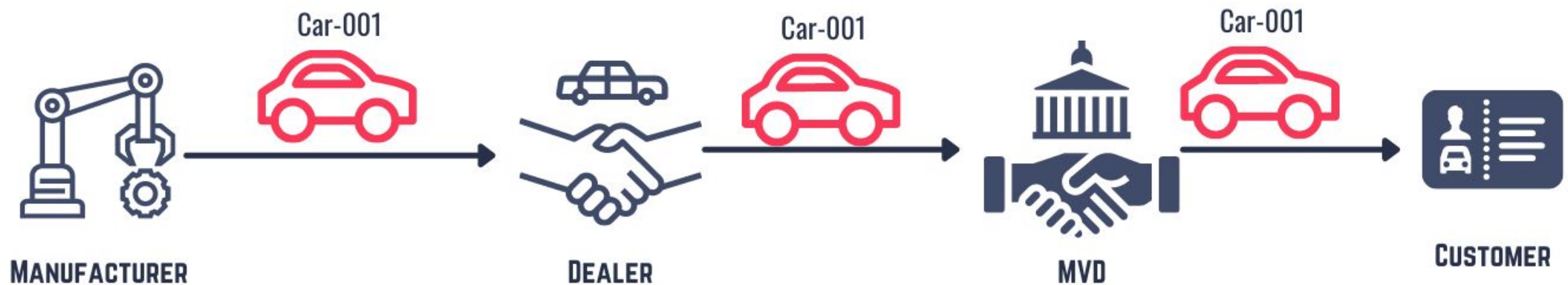
1. List matching orders

❖ Here we obtain all the orders matching to a car

GetMatchingOrders(ctx, carId)

2. Assigning to dealer

❖ Here we assign a car to the dealer based on the order

❖ The order is removed from the private data collection

MatchOrder(ctx, carId, orderId)

3. Registering Car

❖ Register a car to the customer
❖ It can be performed by the MVD only
❖ Status and ownedBy get updated
❖ Car gets registered by a registration number

RegisterCar(ctx, carId, ownerName, registrationNumber)

THANK YOU