A Database Design Methodology

Area of application:

Design of database with its applications.

Perspective:

The method assumes that the primary purpose of the future system is to automate current or planned activities of the enterprise.
The method assumes (as do all database design methodologies) that different views on the enterprise, conflicts, and political differences will be resolved during the database design process.

Life-Cycle:

The **life-cycle** comprises the following steps:

Project Progress Report: Phase I

- I.1. Environment & Requirement Analysis
- I.2. System Analysis & Specification

Project Progress Report: Phase II

- II.1. Conceptual Modeling
- II.2. Logical Modeling
- II.3. Task Emulation
- II.4. Optimization (NOT REQUIRED for the 424 project)

Project Progress Report: Phase III

- III. Implementation
- III.1 Convert Emulated tasks to code
- III.2 Bulk-Loading & Tuning (LIMITED for the 424 project)
- III.3 Testing

Limitation:

The methodology does not cover implementation, testing, maintenance, and project management.

Example

Design a Merryland State Motor Vehicle Administration (MVA) Information System

The system must support all activities of the Merryland State MVA.

The tasks of the MVA include maintaining all information pertaining to: vehicle emissions, *license maintenance*, vehicle registration maintenance, fine and fee maintenance, driving record maintenance and report generation.

Etc., etc., ...

I.1. Environment & Requirements Analysis

The purpose of this phase is to investigate the **information needs** of and the **activities** within the enterprise and determine the **boundary** of the design problem (not necessarily identical to the boundary of the future computerized system, if any).

Input:

Information describing the current status of the enterprise, possible inefficiencies, plans for the future, and constraints that have to be satisfied in conducting business.

Output:

A **Top-Level Information Flow Diagram** describing the major documents and functions, and the boundary of the design problem. **The documents** include the major input, output, and internal documents. **The functions** model the major activities within the enterprise.

Function:

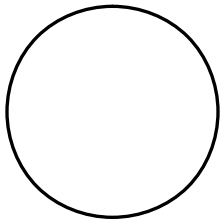
To collect the information about the enterprise and design the top-level information flow diagram.

Guidelines:

- **Techniques:** collect information by contacting interviews of people at all levels of the organization; analyze questionnaires; review short and long term plans, business manuals, files, forms, etc.
  Tools: express a top-level information flow diagram to capture the functions and important documents of the enterprise, and to start the design with the i/o documents and work from the outside in towards a "top-level" design.

  The tool we use for designing the top-level information flow diagram is the following **graphic formalism** for representing **structures and processes**:

structure

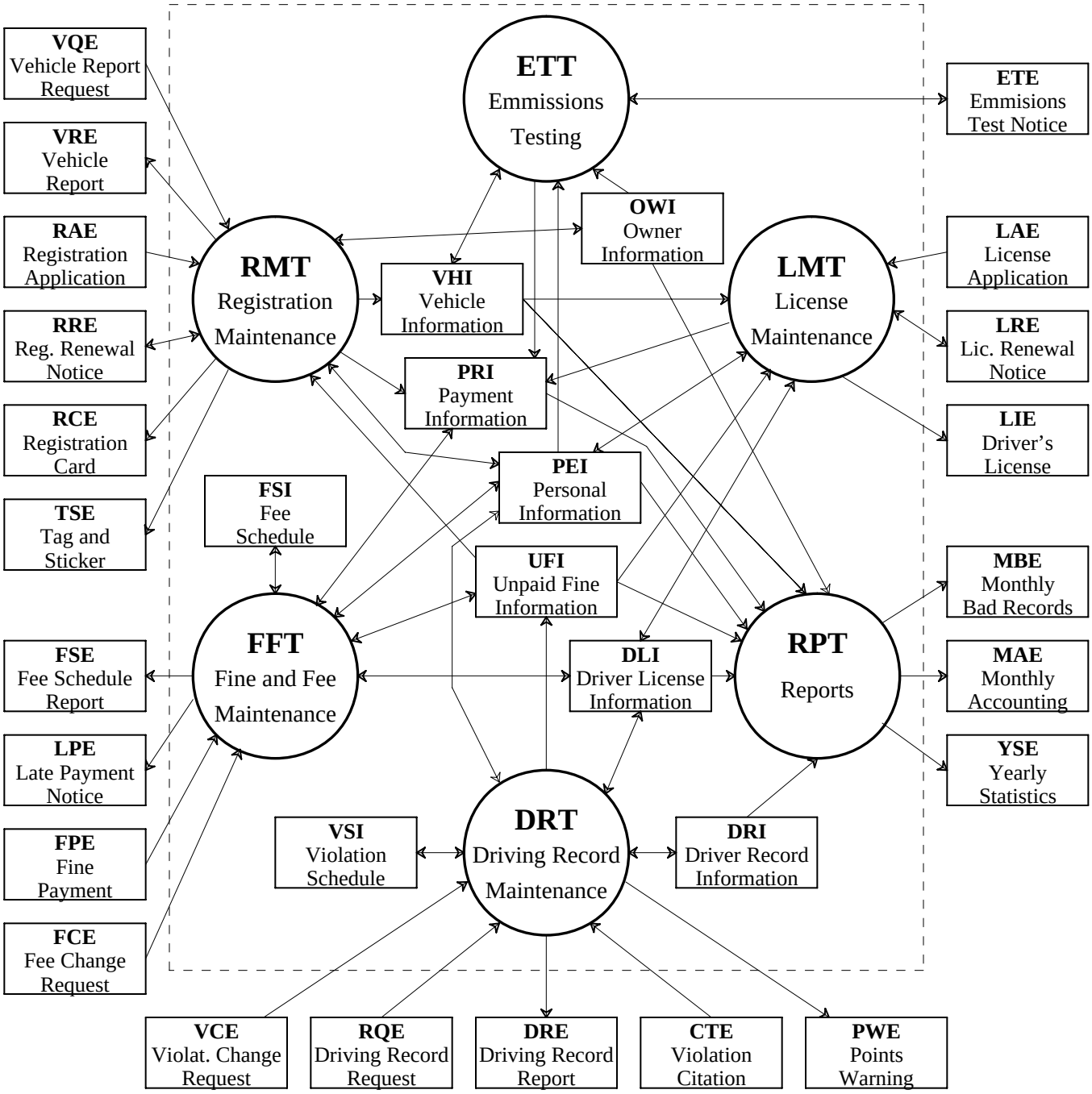


process

information flow

- Two structures are **never** directly connected.
- Two processes are **never** directly connected.

MVA Top-Level Information Flow Diagram



I.2. System Analysis & Specification

The purpose is to divide the functions from the Top-Level Information Flow Diagram hierarchically into tasks. The tasks should be reasonably independent to minimize the task-to-task interfaces (documents).
During the division process, the documents used by each function are also broken down. The process is continued until each task is small enough to be clearly understood, and until each document can be conveniently expressed in terms of data elements that cannot be further divided.
The result is a detailed Task Flow Diagram and a set of forms describing the documents and the tasks.

Input:

The Top-Level Information Flow Diagram and information about the documents and functions from phase 1.

Output:

Task Forms; Document Forms; Document and Data Usage Matrices; and, the detailed Task Flow Diagram.

Function:

Decompose functions and documents. Specify the resulting Task and component Document Forms. Specify Document and Data Usage Matrices. Design detailed Task Flow Diagram.

Guidelines:

- **Technique:** top-down hierarchical decomposition.
- **Tools:** Task Forms; Document Forms; Usage Matrices; and the graphical formalism for Task Flow Diagrams.

Examples of Task Forms

TASK NUMBER:	LMT
TASK NAME:	License Maintenance
PERFORMER:	Driver registration staff
PURPOSE:	Maintain all driver licensing information
ENABLING COND:	Receipt of License Application (LAE); Receipt of License Renewal Notice (LRE); First day of each month; Last day of each month;
DESCRIPTION:	Issue new licenses and process renewals. Print renewal notices on a monthly basis. Delete expired licenses on a monthly basis.
FREQUENCY:	See subtasks (different for each subtask!)
DURATION:	See subtasks (different for each subtask!)
IMPORTANCE:	See subtasks (different for each subtask!)
MAXIMUM DELAY:	See subtasks (different for each subtask!)
INPUT:	(DLI) Driver License Infor; (FSI) Fee Schedule; (LAE) License Application; (LRE) License Renewal Notice; (PEI) Person Info
OUTPUT:	(DLI) Driver License Info; (ENE) Expiration Notice; (LIE) Driver's License; (LRE) License Renewal Notice; (PEI) Person Info; (PRI) Payment Record
DOCUMENT USE:	(DLI) all columns; (LIE) all columns; (LAE) all columns; (PEI) all columns; (FSI) all columns; (PRI) all columns; (LRE) all columns; (ENE) all columns
OPS PERFORMED:	See subtasks
SUBTASKS:	(LMT.1) NEW-LICENSE; (LMT.2) RENEW-LICENSE; (LMT.3) PRINT-RENEWAL-NOTICES; (LMT.4) DELETE-EXPIRATIONS
ERROR COND:	See subtasks

Examples of Task Forms

TASK NUMBER:	LMT.1
TASK NAME:	NEW-LICENSE
PERFORMER:	Driver Registration Staff
PURPOSE:	Issue a license to a previously unlicensed driver, or to a driver whose license has expired.
ENABLING COND:	Receipt of License Application (LAE)
DESCRIPTION:	Checks if the applicant has unpaid fines or untested cars; if not, records the information pertaining to the applicant and prints a driver's license
FREQUENCY:	300,000 per year
DURATION:	Short
IMPORTANCE:	Medium/high
MAXIMUM DELAY:	10 minutes
INPUT:	(LAE) License Application; (FSI) Fee Schedule; (PEI) Person Info; (DLI) Driver License Info
OUTPUT:	(LIE) Driver's License; (DLI) Driver License Info; (PEI) Person Info
DOCUMENT USE:	(DLI) all columns; (LIE) all columns; (LAE) all columns; (PEI) all columns; (FSI) all columns
OPS PERFORMED:	(1) If no PEI tuple exists where PEI.SSN = LAE.SSN, skip to step (2). Otherwise, if PEI.FINE-TEST-STATUS = NOT-OK, stop (person has unpaid fines or untested cars). Otherwise, delete the tuple from PEI. (A new one with the info from LAE will be created - this allows a person to change his/her address and/or name).

(2) Create a new tuple in PEI by copying the following columns from LAE to the PEI tuple: SSN, NAME, ADDRESS, HEIGHT, WEIGHT, SEX, BIRTHDATE. Set PEI.FINE-TEST-STATUS to OK.

(3) If a DLI tuple exists where DLI.DRIVER-ID = LAE.SSN then stop (driver is already licensed).

(4) Create a new tuple in DLI as follows: Copy LAE.SSN to DLI.DRIVER-ID. Set LICENSE-STATUS to OK, TOTAL-POINTS to zero, EXPIRATION-FLAG to "Not Expired", EXPIRATION-DATE to LAE.CURRENT-DATE plus 4 years.

(5) Issue the actual license as follows: activate ISSUE-LICENSE subtask, with parameter DRIVER-ID = DLI.DRIVER-ID.

(6) Retrieve FSI tuple where FSI.PAYMENT-TYPE = "New License". Activate the record payment subtask with parameters: PAYMENT-TYPE = "New License", AMOUNT = FSI.AMOUNT, DATE-RECEIVED = LAE.CURRENT-DATE.

SUBTASKS: (LMT.1.1) ISSUE-LICENSE; (LMT.1.2) RECORD-LICENSE-PAYMENT;

ERROR COND: See (1) and (3) under OPS PERFORMED.

Rule of Thumb for Task Decomposition

- Many performers are required to carry out the task and each performer has different skills, or each can carry out a part independently.
- Different levels of authorization exist for carrying out different parts of the task.
- Different enabling conditions activate parts of the task.
- Different frequencies and durations apply to different parts of the task.
- Input documents are not used uniformly within the task.
- Different documents are used for different parts of the task.
- Many diversified operations are carried out within the task.
- Many subtasks are controlled by the task.

Examples of Task Forms

TASK NUMBER:	LMT.1.1
TASK NAME:	ISSUE-LICENSE
PERFORMER:	Driver Registration Staff
PURPOSE:	Print a license to be issued to a driver (either a new license or a renewal).
ENABLING COND:	Activated by LMT.1 or LMT.2
DESCRIPTION:	Collect driver information for the specified driver
FREQUENCY:	1,050,000 per year
DURATION:	Very short
IMPORTANCE:	Medium/high
MAXIMUM DELAY:	10 minutes
INPUT:	(DLI) Driver License Info; (PEI) Person Info
OUTPUT:	(LIE) Driver's License
DOCUMENT USE:	(LIE) all columns; (PEI) all columns; (DLI) DRIVER-ID, EXPIRATION-DATE, LICENSE-CLASS
OPS PERFORMED:	(1) Join DLI.DRIVER-ID to PEI.SSN; select those tuples where DLI.DRIVER-ID = DRIVER-ID (input parameter). This will select one tuple, since DLI.DRIVER-ID is the key. (2) Copy all relevant retrieved information to a new LIE (i.e., print a driver's license using the information)
SUBTASKS:	none
ERROR COND:	No DLI/PEI tuple exists for the given LIC.DRIVER-ID.

Example of Document Forms

DLI: Driver-License-Information
- Driver-ID
- License-Class
- Issue-Date
- Expiration-Date
- License-Status (Ok, Warning, Suspend, Expired)

FSI: Fee Schedule
- Payment-Type
- Amount

PRI: Payment-Record
- Payment#
- Payment-type
- Amount
- Date-Received

PEI: Person-Information
- SSN
- Name
- Height
- Weight
- Sex
- Birthdate
- Address
- Fine-Test-Status

LAE: License-Application
- Driver-ID (i.e. appliant's SSN)
- Name
- Address
- Height
- Weight
- Sex
- Birthdate
- License-Class
- Current-Date
- Payment-Amount

LRE: License-Renewal-Notice
- Driver-ID
- Name
- Address
- Expiration-Date
- Payment-Amount
- Current-Date

ENE: Expiration-Notice
- Driver-ID
- Name
- Address
- Expiration-Date

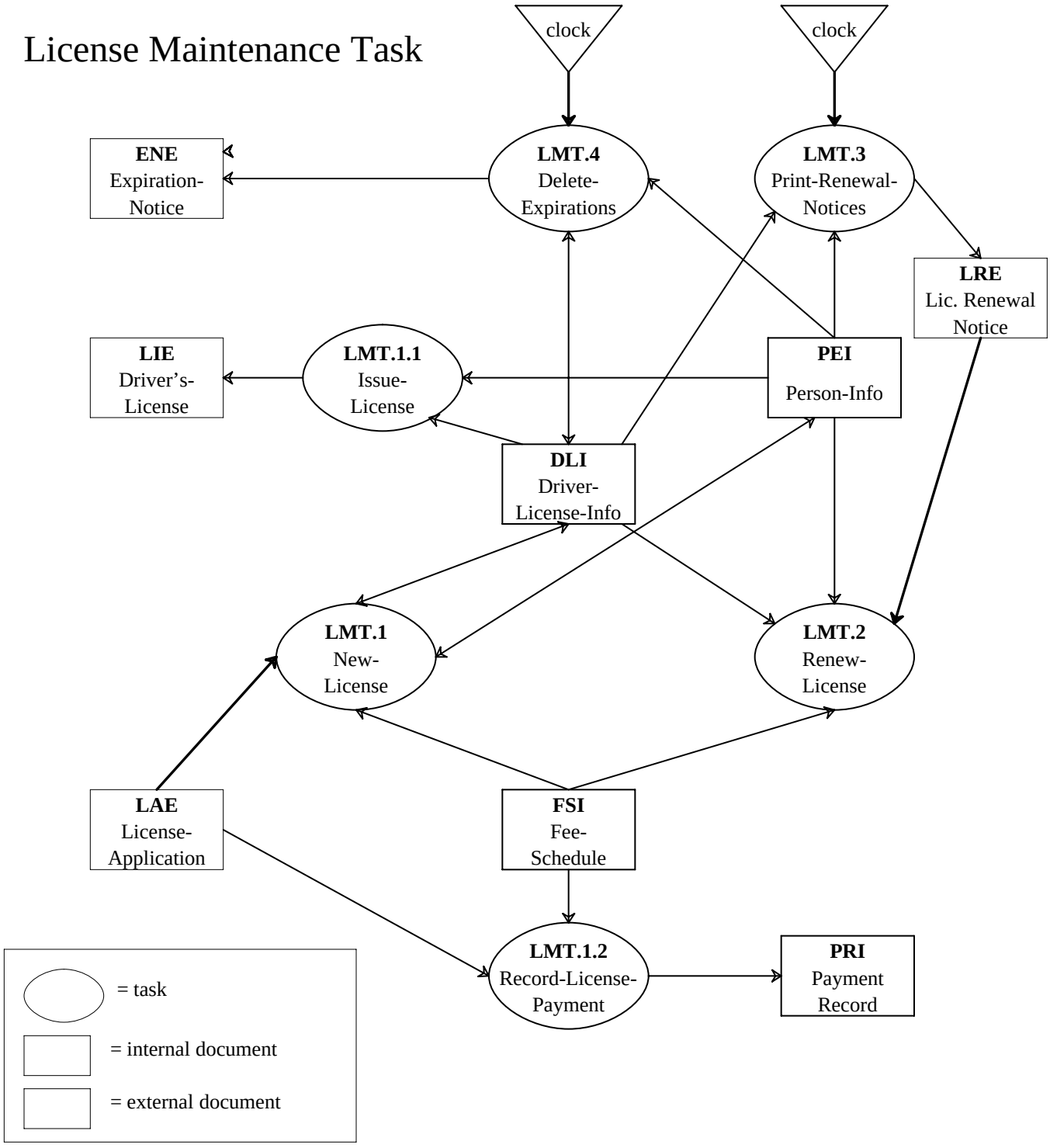
LIE: Drivers-License
- Driver-ID
- Name
- License-Class
- Height
- Weight
- Sex
- Expiration-Date
- Birthdate
- Address

Example of Task-Data Usage Matrices

TASK / DLI Field	LMT.1	LMT.1.1	LMT.1.2	LMT.2	LMT.3	LMT.4
Driver-ID	I/O	I		I	I	I
License-Class	O					
Issue-Date	O					
Expiration-Date	O			I	I	I
License-Status	O			I		

TASK / PEI Field	LMT.1	LMT.1.1	LMT.1.2	LMT.2	LMT.3	LMT.4
SSN	I/O	I		I	I	I
Name	I/O	I			I	I
Height	I/O	I				
Weight	I/O	I				
Sex	I/O	I				
Birthdate	I/O	I				
Address	I/O	I			I	I
Fine-Test-Status	I/O	I		I		

Example of Task Flow Diagram



II.1. Conceptual Modeling

The purpose of this phase is to design a **conceptual schema** of the database. We will use the E-R data model.

Input:

The Document Forms and the Task-Data Usage Matrix produced in the previous phase.

Output:

A Conceptual Schema described in terms of the E-R data model.

Function:

To design the Conceptual Schema from the Document Forms and the Task-Data Usage Matrix.

Guidelines:

- **Techniques** for conceptual schema design. E.g. **semantic data modeling and normalization!**
- The **tool** is the graphical formalism for the DDL of the data model.

II.2. Logical Modeling

The purpose of this phase is to map the **conceptual schema** of the database to a logical data model that is supported by (directly implementable on) a DBMS. We will use the relational model as a logical data model.

Input:

The E-R diagrams and Functional Dependencies discovered by the 1-1, 1-many, and other constraints imposed by the semantic of the application.

Output:

A Logical Schema (relational in our case).

Function:

Map the E-R model to tables, their keys, and FDs. Then normalize the relations to obtain as close to BCNF relations.

Guidelines:

- **Technique:** use the E-R to table mapping technique and the key algorithms.

Normalization !!!

- Make sure all domains are atomic (1. NF)
- Identify all Functional Dependencies (FDs)
- Decompose relations to remove nonfull FDs (2. NF)
- Decompose relations to remove transitive FDs (3. NF)
- Make sure all determinants are candidate keys (BCNF)

Example of Conceptual Schema

DLI: Driver-License-Information

Driver-ID	Issue-Date	Expiration-Date	License-Class	License-Status

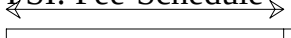
PEI: Person-Information

SSN	Name	Height	Weight	Sex	Birthdate	Address	Fine-Test-Status

PRI: Payment-Record

Payment#	Payment-Type	Amount	Date-Received

FSI: Fee-Schedule



Payment-Type	Amount

No Normalization Needed

- All domains are atomic (1. NF)
- No nonfull FDs (2. NF)
- No transitive FDs (3. NF)
- All determinants are candidate keys (BCNF)

II.3. Task Emulation

The purpose of this phase is to obtain the design and specification of the software that performs the tasks **before** any database implementation starts. In other words, before creating a schema in the DBMS, the application programming is fully specified. This gives the opportunity to correct the logical schema when it is incomplete, superfluous, or even dead wrong. Doing the design of the database schema and the applications using it simultaneously complements these two orthogonal specifications and catches most of the errors before the implementation.

Input:

The Logical Schema from the previous phase and the Task Forms.

Output:

- The set of design specifications of the pieces of software that performs the tasks described in the task forms. The design specifications can be given in terms of abstract programs with embedded sequences of DML statements, using the DML of the data model.
- External Schemata based on the Logical Access Path Schema.
- Internal Schema constituting an optimized Logical Access Path Schema.

Function:

- Use the Task Forms describing the tasks. Formulate for each task an abstract program including embedded sequences of DML statements that perform the task using the conceptual schema. (During this phase small corrections of the conceptual schema may be needed to support the tasks: *validation*).
- Design the Logical Access Paths for all queries. Construct the Task-Query Frequency and Probability Matrices. Integrate the Query Graphs into the Logical Access Path Schema.
- Group Conceptual Schema relations and Views (nodes in the Logical Access Path Schema) used by a task or a set of similar tasks into External Schemata. Where applicable, replace sequences of embedded DML statements in the task specifications by statements accessing the views in the External Schemata.
- Design the Internal Schema based on the Logical Access Path Schema (by introducing indices for the most frequently use Access Paths).

Guidelines:

- **Techniques:** those that apply to the use of the particular DML.
- The **tool** are the DML, the graphic notation for Access Paths, the Task-Query Frequency and Probability Matrices, and the Internal Schema Definition Language (i-DDL).

Task Design Specification

```
LMT-1-NEW-LICENSE
   {Read license application data from LAE document into appropriate variables}
{ (1) See if applicant's personal info is already in db }
EXEC SQL SELECT FINE-TEST-STATUS INTO :FINE-TEST-STATUS
   FROM PEI WHERE SSN = :SSN ;
if SQLCODE = found then
   if FINE-TEST-STATUS <> "OK" then
      print-message("person has unpaid fines or untested cars")
   else   { delete old info }
      EXEC SQL DELETE FROM PEI WHERE SSN = :SSN
   endif
{ (2) Insert person into db }
EXEC SQL INSERT
   INTO PEI (SSN, NAME, BIRTHDATE, ADDRESS, SEX, HEIGHT, WEIGHT,
                FINE-TEST-STATUS)
   VALUES (:SSN, :NAME, :BIRTHDATE, :ADDRESS,
              :SEX, :HEIGHT, :WEIGHT, "OK") ;
{ (3) Check if person is already licensed. }
EXEC SQL SELECT DRIVER-ID FROM DLI WHERE DRIVER-ID = :SSN ;
if SQLCODE = found then
   print-message("driver already licensed")
   return
endif
{ (4) Create new DLI tuple }
EXEC SQL INSERT
   INTO DLI (DRIVER-ID, LICENSE-CLASS, ISSUE-DATE, EXPIRATION-DATE,
                LICENSE-STATUS, EXPIRATION-FLAG)
   VALUES (:SSN, LICENSE-CLASS, :DATE, :DATE + 4 years,
              "OK", "Not Expired") ;
{ (5) Issue License }
call LMT-1-1-ISSUE-LICENSE
{ (6) Record Payment }
EXEC SQL SELECT AMOUNT INTO :AMOUNT FROM FSI
   WHERE PAYMENT-TYPE = "New License" ;
if SQLCODE = found then
   PAYMENT-TYPE <- "New License";
   call LMT-1-2-RECORD-LICENSE-PAYMENT
endif
end LMT-1-NEW-LICENSE.
```

II.4. Optimization

The purpose of this phase is to find commonalities in the access of the tasks and improve performance on the most frequent ones. All this information is integrated in what is termed **Logical Access Path (LAP)** schema.

Input:

The queries from the tasks, their frequencies and their weights.

Output:

The LAP schema.

Function:

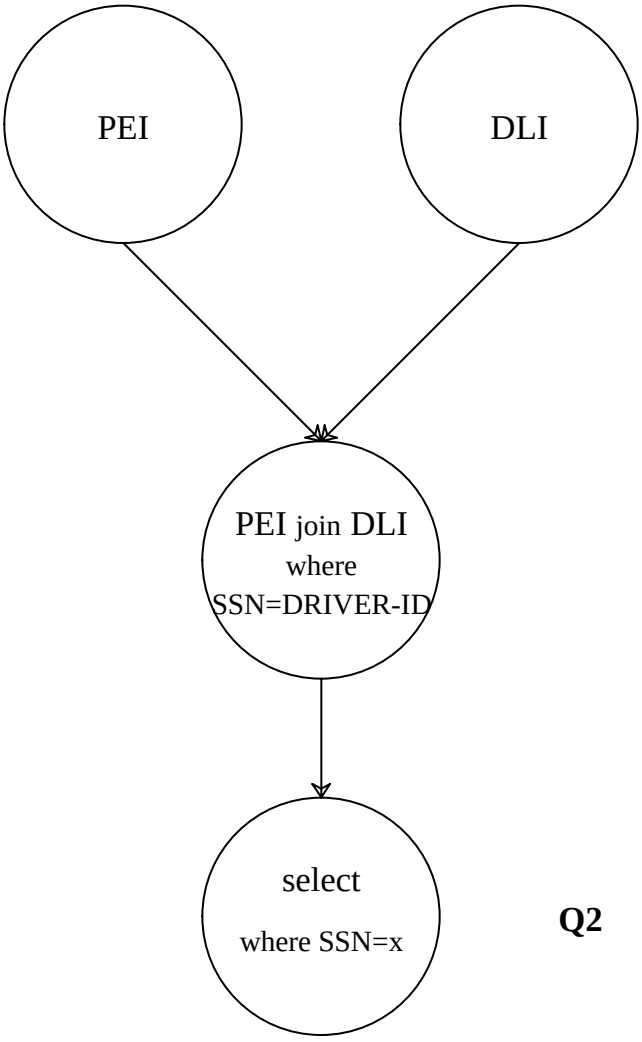
Integrate individual query graphs into the LAP schema and compute their probabilities.

Guidelines:

- **Technique:** query (view) integration algorithms.

Example of Query Graph

```
LMT-1-1-ISSUE-LICENSE
    EXEC SQL SELECT PEI.NAME, PEI.ADDRESS, DLI.LICENSE-CLASS,
                    PEI.HEIGHT, PEI.WEIGHT, PEI.SEX, DLI.EXPIRATION-DATE,
                    PEI.BIRTHDATE, PEI.ADDRESS
                INTO :NAME, :ADDRESS, :LICENSE-CLASS, :HEIGHT, :WEIGHT,
                        :SEX, :EXPIRATION-DATE, :BIRTHDATE, :ADDRESS
                FROM PEI, DLI
                WHERE PEI.SSN = DLI.DRIVER-ID AND
                            PEI.SSN = :DRIVER-ID
```



Q2

External Schemata

- External schemata are defined to fit the applications' queries using the nodes in the Logical Access Path.

EXAMPLE

- PEI join DLI where PEI.SSN = DLI.DRIVER-ID is used in Q2 and Q3, with combined execution probability of 55%.

```
DEFINE VIEW LICENSES AS
    SELECT NAME, ADDRESS, LICENSE-CLASS, HEIGHT,
        WEIGHT, SEX, EXPIRATION-DATE, BIRTH-DATE
    FROM PEI, DLI
    WHERE PEI.SSN = DLI.DRIVER-ID;
```

- Revised LMT.1.1 using the view LICENSES:

```
LMT-1-1-ISSUE-LICENSE
    EXEC SQL SELECT *
        INTO :NAME, :ADDRESS, :LICENSE-CLASS, :HEIGHT, :WEIGHT,
            :SEX, :EXPIRATION-DATE, :BIRTHDATE
        FROM LICENSES
        WHERE SSN = :DRIVER-ID;
    {print the information on LIE}
```

Task Query Probability Matrix

TASK / QUERY	LMT.1	LMT.2	LMT.3	LMT.4	Prob
Q1	1	0	0	0	0.07
Q2	1	2	0	0	0.55
Q3	0	0	1	1	˜0
Q4	1	0	0	0	0.07
Q5	1	1	0	0	0.31
SUM	4	3	1	1	
freq. (per year)	300K	750K	12	12	1.05M

Task Update Probability Matrix

TASK / REL.	LMT.1	LMT.2	LMT.3	LMT.4	Prob
PEI	2	0		0	0.14
DLI	1	1		1	0.43
PRI	1	1		0	0.43
SUM	4	2		1	
freq. (per year)	300K	750K		12	1.05M

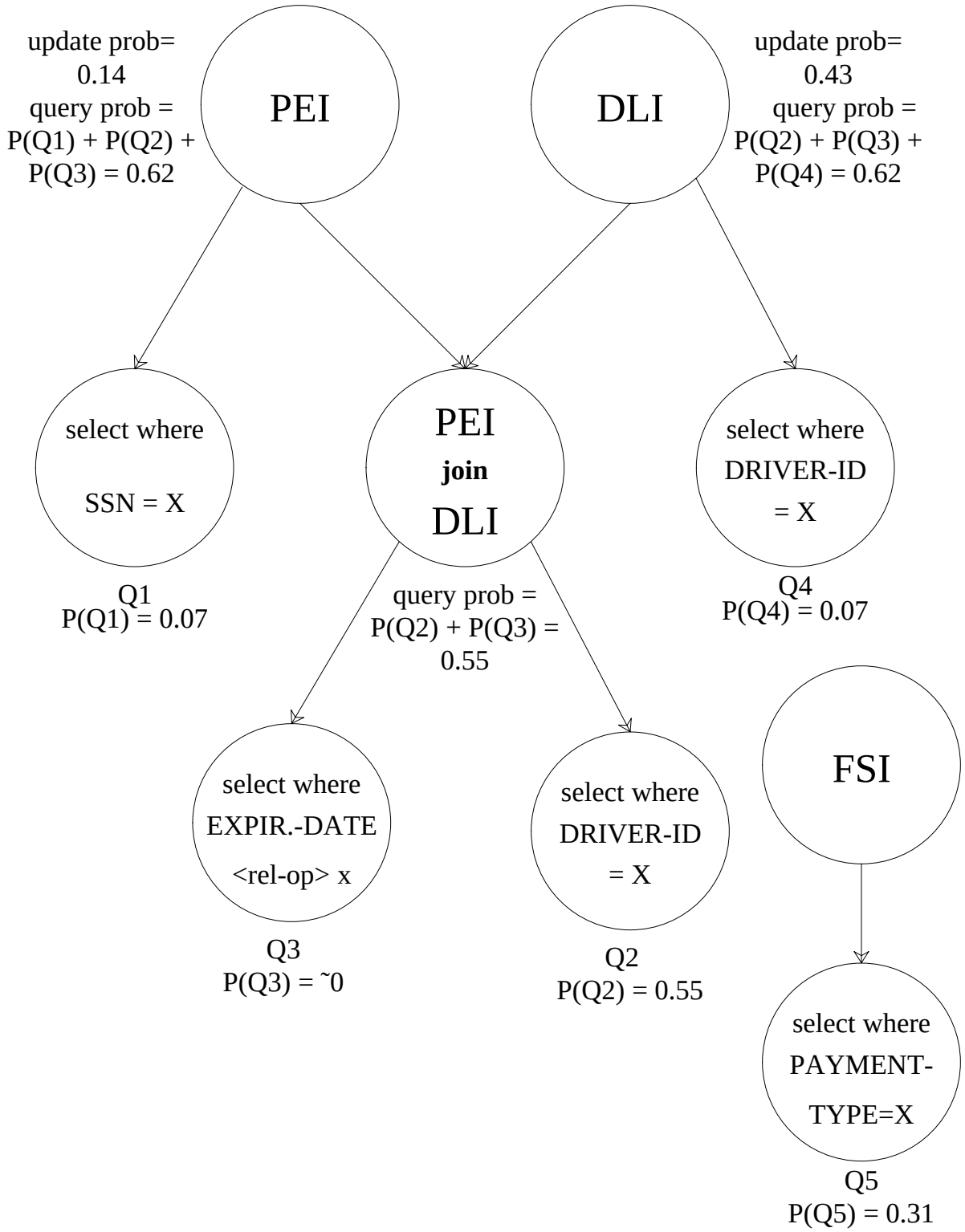
Computation of Probabilities

$T = \Sigma_j freq_j$ = 1.05 M

A_{ij} = matrix entry for row i, column j

$Prob_i := \Sigma_j (A_{ij}/SUM_j) * (freq_j / T)$

Example of Logical Access Path



Indexing

Secondary indices are introduced on a base relation if:

- there is a high probability for queries against it
- the queries against it are not on the primary key
- there is a low probability for updates against it

EXAMPLE: No Indices Introduced

- All Queries (except Q3) are selections on the value of a relation's primary key.
- Possible secondary index on attribute, DLI.Expiration-Date, used in Q3:
- Probability of performing Q3 = ˜0
- Probability of update to DLI is high
- In this case, secondary index doesn't provide much benefit.

III. Implementation

III. Convert Emulated tasks to code

(Not part of the methodology!)

The purpose of this phase is to translate the conceptual schema and the task design specifications into actual schema definitions and application program modules.

Input:

The graphic version of the conceptual, external, and internal schemata and the task designs.

Output:

A textual version of the input with the schemata defined in terms of the DDL and the tasks programmed in terms of the host-language with embedded DML statements.

Function:

To translate the graphic version of the schemata into a definition of the schemata using the DDLs. To translate the task designs into host-language modules.

Guidelines:

- **Technique:**. Not really. The translation is very simple.
- **Tools:** the DDL and the host-language with the embedded DML statements.

III.2 Bulk Loading and Tuning

(Not part of the methodology!)

The purpose of this phase is to load the real stuff and fine tune its performance.

Input:

The schema definitions and the application programs from the previous phase. A set of test data.

Output:

The database system.

Function:

Almost always this is very painful step which can take several weeks or even months. The biggest problem is data errors that need to be cleaned before entered. Bulk loading implies high volume of data (unlike your CMSC 424 project).

Guidelines:

- **Technique:** patience!
- **Tool:** bulk loaders and scripting languages.

III.3 Testing

(Not part of the methodology!)

The purpose of this phase is to test the schema definitions and the tasks.

Input:

The schema definitions and the application programs from the previous phase. A set of test data.

Output:

The tested program.

Function:

Test all possible paths (if you can count them!). Then, test it again.

Guidelines:

- **Technique:** plan and structure the test to make sure that all possibilities are tested.
- **Tool:** the DBMS itself.

DON'T FORGET

- The secret behind successful Database Design is careful analysis, specification, and design. These are done in the phases I.1-II.4 of the methodology. Having done a careful analysis on these, it would give enough chances not to fail!

- There are always bugs in large databases. Careful testing eliminates only the most obvious. Testing requires a systematic methodology different than the one used by Microsoft!
- Large databases are used for many years. Maintaining a database throughout its life-time typically takes several times more than development. It is impossible to maintain a database with an undocumented design. The documents produced by this methodology is the design specification and will be the heart of the documentation if properly maintained. Without the methodology, there is no common language to exchange design specifications.