ADVANCED DATABASE SYSTEMS

COURSE NAME: MTECH – IST SEMESTER

COURSE CODE: CSE511

Teacher Incharge: *Dr. Shifaa Basharat*
Contact: *fazilishifaa@gmail.com*

The Temporal Data Model

A "data model" consists of two components, namely a set of objects and a language for querying those objects. In a temporal data model the objects vary over time, and the operations in some sense "know" about time. In this unit, we describe a Semantic Temporal model based on the Extended Entity-Relationship model *(STEER),* which distinguishes between conceptual and temporal objects. A *conceptual object*, once it is created, can always be referenced at any future time, whereas a *temporal object*, which we call an *entity role,* has a specific existence lifespan. For example, information concerning a *STUDENT* conceptual object can be referenced even after the student has completed his studies. However, the role of that entity as an *ENROLLED-STUDENT* has specific start and end times that define its lifespan. *(STUDENT* is the owner entity of *ENROLLEDSTUDENT* role.)

The *STEER* model characterizes the properties of entities (conceptual objects), entity roles (temporal objects), and (temporal and non-temporal) attributes. It also defines temporal constraints among entity roles, differentiates between temporal and conceptual relationships, and provides rules for preserving temporal integrity constraints.

We complement our model by providing temporal query language constructs. The temporal query language distinguishes between temporal and conceptual objects/relationships. It allows selection conditions to retrieve attributes and relationships of a role or an entity type, since attributes and relationships of a role type and its owner entity type are public to each other and can be inherited. It also provides natural and high level temporal element constructor operators that simplify temporal query expressions. Finally, it supports temporal version restriction operators and allows multiple temporal scopes in a temporal projection.

The main features of the STEER model have been discussed in the following sections.

1. **Conceptual Objects: Entities**
   Our goal is to define guidelines for determining the basic aspects of an object life time. The conceptual existence of an object does not directly correspond to the birth, death, and change of the object. Objects need to be modeled in a mini-world when they become of interest. For example, employees exist in the real world as persons. However, they become objects of interest to a company only when the company wants to hire them. At this point, the company may still want to record previous information about these persons. If an employee leaves the company, the employee remains an object of interest as long as the company still wishes.
   Each *conceptual entity e* has an *existence time,* denoted by *ET,* which is unrelated to the concept of lifespan. The *start time point ST* of the existence time refers to the time the concept of the entity is materialized. There is no *end time point* of an existence time. The end time can be considered to be infinity in this model, because a concept (an entity) once realized never ceases to exist. The only time that characterizes an entity is the start

time of its existence. Hence, $ET = [ST, \infty)$. (We also use the notation $T(e)$ to refer to the existence time of an entity e.)

There are two important ramifications in associating existence time with entities:

- We can define and treat future planning concepts using similar mechanisms to those used for historical concepts.
- We can enhance the power of query languages and simplify their constructs while dealing with conceptual objects, by using start time point of existence time as the earliest possible time the entity can be referenced.

An *entity type* is a set of entities of the same type; that is, entities that share the same properties. An entity type is diagrammatically represented by a rectangular box (see Figure 1).

2. Temporal Objects: Roles

Entities describe one aspect of the real world, the conceptual one. The other aspect is captured by temporal objects. The classification of objects as temporal and conceptual gives our model the capability to faithfully represent the way people perceive the real world. Temporal objects materialize the active role that conceptual objects play in the temporal dimension.

A temporal object is called an *entity role*, since it represents the time that the entity is participating in that role. A *role type* is a set of entity roles of the same type; that is, roles that share the same properties. Each role type is associated with a single entity type called its *owner entity*. Hence, *owner(entity role) = entity r role(entity) = entity role*. A role type is diagrammatically represented by a dotted rectangular box, and connected to an owner entity (see Figure 1). Each entity role *ro* of a role type *RO* is associated with a temporal element $T(ro)\ C\ [t_o, \infty)$ which gives the lifespan *LS* of the role.

The following general set of rules must hold on roles:

- Start time of the lifespan of an entity role must be greater or equal to the start time of the existence time of the (conceptual) owner entity. This implies a top down approach in creation of role types; that is, before a role is created its corresponding (owner) entity must exist.
- A role type is restricted exactly to one owner entity type.
- A role type can have only temporal attributes.
- (Temporal) attributes of a role type are *public* to the owner entity type; that is, an owner entity refers to these attributes as though they are attributes of the owner entity.
- Similarly, (temporal and non-temporal) attributes of an entity type are public to all associated role types.
- A role can access all relationship instances for relationship types in which the

owner entity participates.

- Similarly, an entity can access all relationship instances for relationship types in which the associated role participates.

3. **Temporal Constraints among Roles**

   STEER model enforces two temporal constraints among roles:

   - *Existence Constraint:* A sup-existence/sub-existence constraint, denoted by *ROi/ROj,* holds between two role types *ROi* and *ROj* iff the following holds:
     $\{\forall\ ro_{jk} \in ROj, \exists\ ro_{il} \in ROi\ such\ that\ ro_{jk} \equiv ro_{il}\}$;

     that is, every entity role *ROj* represents the same entity role in *ROi.* The existence constraint implies a top-down approach in the creation of roles. A member role of a sub-existence represents the same real world entity as some member of the sup-existence. An entity role cannot exist in the database merely by being a member of a subexistence; it must be also a member of the sup-existence.

   - *Li]espan Constraint:* A sup-lifespan/sub-lifespan constraint, denoted by *ROi/ROj,* holds between two role types RO_i and *ROj* iff the lifespan of any entity role $ro_{jk} \in ROj$ is a subset of the lifespan of the entity role $ro_{il} \in ROi$ where $ro_{jk} \equiv ro_{il}$; that is, $T(ro_{jk}) \subset T(ro_{il})$. Notice that the lifespan constraint implies the existence constraint, but not vice versa.

4. **Non-Temporal Attributes**
   Attributes are properties of objects. Non-temporal attributes can be only properties of conceptual entity types but not of role types. The value of a non-temporal attribute of an entity holds over the entire existence time of the entity. *(Same as properties of the non-temporal attributes of the ER model—Check yourself)*

5. **Temporal Attributes**
   Each entity type E_i (role type RO_i) may have a set of basic temporal attributes TA_{il}, Ta_{i2}, ..., TA_{in}, and each temporal attribute TA_{ij} is associated with a domain of values $dom(TA_{ij})$. For example, a temporal attributes of the PERSON entity type is Name and a non-temporal attribute is SSN (see Figure 1).
   For roles, the temporal value of each attribute TA_i of ro, which we refer to as $TA_i(ro)$, is a partial function $TA_i(ro) : T(ro) \rightarrow dom(TA_i)$. The subset of T(ro) in which $TA_i(ro)$ is defined is denoted by $T(TA_i(ro))$. It is assumed that TA_i has a NULL (or UNKNOWN) value during the intervals $T(ro) - T(TA_i(ro))$.

In the case of entities, the temporal value of each attribute TA_i of e, which we refer to as $TA_i(e)$, is a partial function TAi(e) : ET(e) $\rightarrow$ dom(TA_i). The subset of ET(e) in which $TA_i(e)$ is defined is denoted by $T(TA_i(e))$. It is assumed that TA_i has a NULL (or UNKNOWN) value during the intervals T(e) - $T(TA_i(e))$.

The partial function that describes the values of a temporal attribute is also called *a temporal assignment*. The subset of chronons (time points) during which a temporal attribute is defined is called the *temporal element of the temporal assignment.*

Several types of temporal attributes exist:

- *A temporal single-valued attributes* has at most a single atomic value for each entity (role) at each time instant [t].
- *A temporal multi-valued attribute* can have more than one value for an entity (a role) at a given time instant [t]; hence, its domain is the power set *P(V)* of some simple domain V.
- *A temporal composite attribute* is a list of several component temporal attributes, and its value for each entity at time instant [t] is a concatenation of the values of its components. The temporal element of a temporal assignment of a composite attribute is the union of the temporal elements of the temporal assignments of its components.

In the STEER model, each entity will be associated with a system-defined non-temporal *SURROGATE* attribute whose value is unique for every entity in the database. The value of this attribute is not visible to users, and is never altered.

6. Conceptual Relationships

A conceptual relationship type R of degree n has n participating entity types E_1, E_2, ... ,E_n. Each *relationship instance* r in R is an n-tuple r =< e_1,e_2,... ,e_n > where each $e_i \in E_i$. Each relationship instance r in R has an existence time *ET.* The start time of the existence time of a relationship instance must be greater or equal to the start time of the existence time of each of the participating entities; that is, $ST(r) \geq ST(e_i)$ *for each* $e_i \in E_i$ *(i = 1, 2,..., n).*

7. Temporal Relationships

Our model supports temporal relationships. A temporal relationship type TR of degree n has n participating entity types (role types) O_1, O_2,..., O_n where O_i is either an entity type ($O_i \equiv E_i$) or a role type ($O_i \equiv RO_i$). Each temporal relationship instance tr in TR is an n-tuple tr =< o_1, o_2,...,o_n > where o_i is either an entity ($o_i \equiv e_i$, $e_i \in E_i$) or an entity role ($o_i \equiv ro_i$, $ro_i \in RO_i$).

Each temporal relationship instance tr in TR is associated with a temporal element T(tr) which gives the lifespan of the temporal relationship instance. If all participating objects are entity roles, then the lifespan of the temporal relationship instance must be a subset of the intersection of the lifespans of the roles; and if all participating objects are entities,

then the start time of the lifespan of the temporal relationship instance must be greater or equal to the start times of all existence times of the entities.

8. Temporal Constraints among Relationships

Our model enforces two constraints on temporal and conceptual relationships:

- *R-existence Constraint:* A sup-R-existence/sub-R-existence constraint, denoted by *R/TR,* holds between a conceptual relationship R and a temporal relationship *TR* where all participating object types are role types iff $\forall\ tr_i = (ro_1, ro_2, ..., ro_n) \in TR$ the following two conditions must be satisfied:
  (a) $\exists\ r_i = (e_1, e_2, ..., e_n) \in R$, such that $owner(ro_j) = e_j$, for j=1,2 . . . . , n.
  (b) The start time of the llfespan of the temporal relationship instance tr_i must be greater or equal to the start time of the existence time of the conceptual relationship r_i.

- *R-lifespan (time order) Constraint:* A sup-R-llfespan/sub-R-lifespan constraint, denoted by *TR/R* holds between a temporal relationship *TR* and a conceptual relationship R where all participating object types are role types iff $\forall\ r_{i=}(e_l, e_2, ..., e_n) \in R$ the following two conditions must be satisfied:
  (a) $\exists\ tr_i = (ro_l, ro_2, . . . , ro_n) \in TR$, such that $owner(ro_j) = e_j$ for all j = 1 , 2 , . . . , n .
  (b) The start time of the existence time of the conceptual relationship instance r_i must be greater or equal to the start time of the lifespan of the temporal relationship tr_i.

The R-existence and R-lifespan constraints are denoted diagrammatically in a similar way to existence and lifespan constraints for role types. Notice that the R-lifespan constraint is, in some sense, the reverse constraint of the lifespan constraint on role types. It is used to model the cases where a conceptual relationship cannot exist until after a temporal relationship has started. For example, students cannot get transcript entry for courses until *after* they have been enrolled.

EXAMPLE:

Consider the example database schema in Figure 1, which describes a simplified organization for part of a *UNIVERSITY database.* The database includes the (conceptual) entity types *PERSON, STUDENT, FACULTY, COURSE,* and *SECTION.* Any entity instance that is a member of any of these entity types is associated with an existence time. The entity types *STUDENT* and *FACULTY are* subtypes of the entity type *PERSON.*
The role types are diagrammatically represented by a dotted rectangular box, and connected to their owner entity types. The role types and their owner entities **are:**

owner(LIVING-PERSON) = PERSON
owner(ENROLLED-STUDENT) = STUDENT
owner(CURRENT-FACULTY) = FACULTY
owner(VALID-COURSE) = COURSE
owner(ACTIVE-SECTION) = SECTION

The conceptual relationship types are:
CS between *COURSE* and *SECTION*
TAUGHT between *FACULTY* and *SECTION*
TRANSCRIPT between *STUDENT* and *SECTION*

The temporal relationship types are:
A CTIVE-CS between *VALID-COURSE* and *ACTIVE-SECTION*
IS-TEA CHING between *C URRENT-FA C ULTY* and *A CTIVE-SECTION*
ENROLLED between *ENROLLED-STUDENT* and *ACTIVE-SECTION*

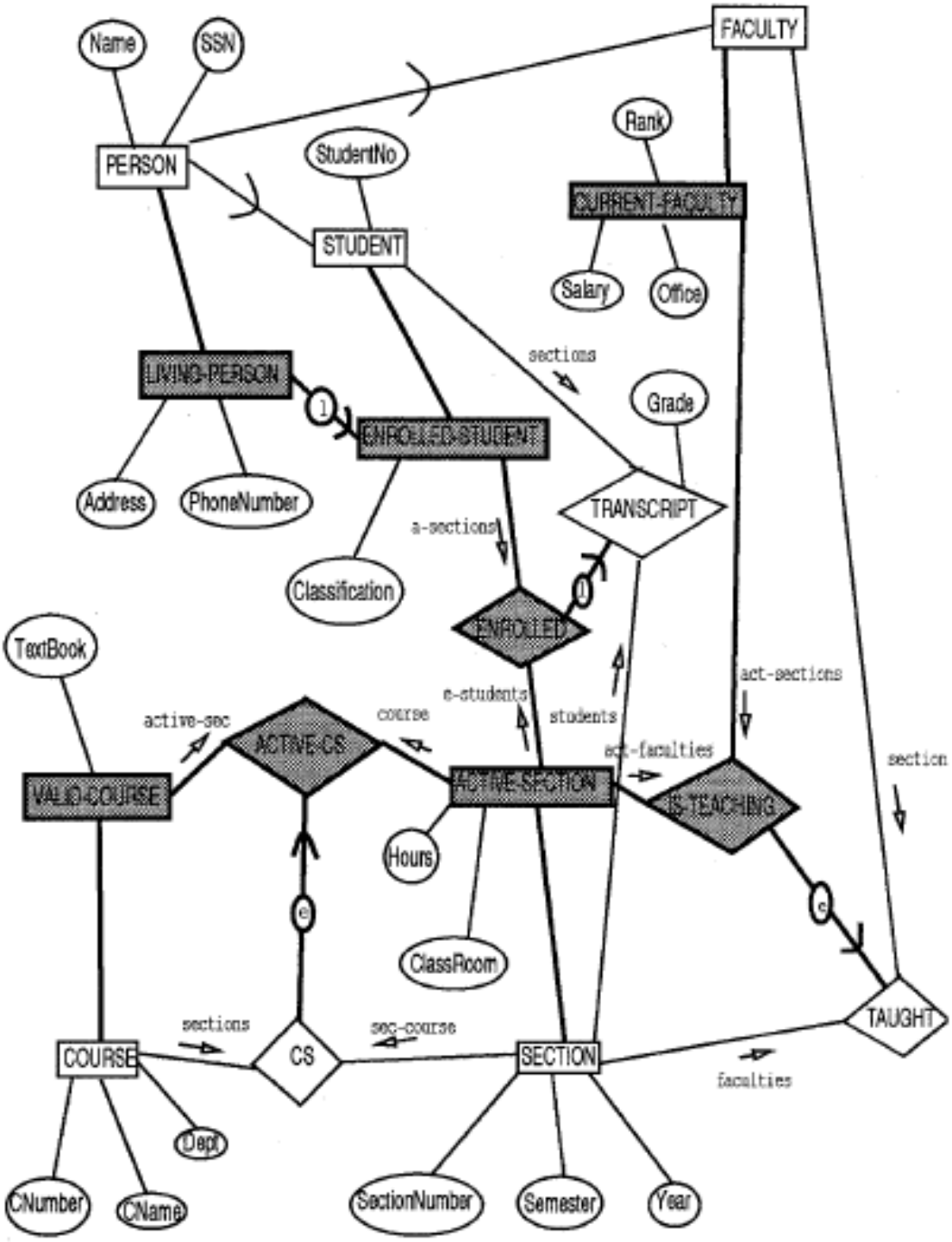

Figure 1: A Temporal EER Schema for part of a University Database

TEMPORAL QUERY LANGUAGE:

Our temporal query language derives its simplicity and expressiveness from the *STEER* data model; in particular from the distinction between temporal and conceptual objects, and temporal and conceptual relationships. The query language used is a temporal extension of *GORDAS* which is a functional query language with two clauses: *GET* and *WHERE*. The *WHERE-clause* specifies conditions for the selection of entities from a *root entity type,* while the *GET-clause* specifies the information to be retrieved for each selected entity. For example, consider the following (non-temporal) *GORDAS* query specified on the database of Figure 1:

Q1. GET < Name, SSN, < CName of sec-course, Semester, Year >
of sections > of STUDENT
WHERE Address of STUDENT = 'Arlington'

Here, the root entity type, specified at the end of the *GET-clause,* is *STUDENT*. The *WHERE-clause* is evaluated individually for each entity in the root entity type, and selects each entity that satisfies the *WHERE-dause.* In this query, each *STUDENT* entity who lives in *'Arlington'* is selected. (Note that the *Address* attribute is visible to *STUDENT* by being inherited from *LIVING-PERSON via PERSON).* The *of STUDENT* in the *WHERE-clause* is optional, and can be left out. For each selected entity, the *GET-clause* retrieves the student *Name, SSN* (both inherited from *PERSON)* and *sections,* and for each of the student's sections the *CName, Semester* and *Year* are retrieved. The connection names such as *sec-course* and *sections* are used to specify related entities of the root entity type in a functional way as though they were attributes of root entities. Hence, the path *sections of STUDENT* specifies the *SECTION* entities related to each *STUDENT* entity via the *TRANSCRIPT* relationship.

- **Temporal Projection**
  A temporal query may involve a temporal selection condition or a temporal projection condition or both. The general philosophy of *GORDAS* is to maintain a clean separation between the specification of conditions for selection of entities (in the *WHERE* clause) and the specification of information to be displayed (in the GET-clause). To maintain this philosophy, we will specify a temporal projection on the data to be displayed at the end of the *GET-clause*. For example, consider the query to retrieve the history of the *Address* and *PhoneNumber* of *'John Smith'* during the period 1985 to 1990:

  Q2: GET < Address, PhoneNumber > of PERSON : [1/1/1985, 12/31/1990]
  WHERE Name = 'John Smith'

  The term *PERSON:* [1/1/1985, 12/31/1990] at the end of the *GET-clause* specifies that the temporal assignment for *'John Smith'* is to be retrieved during the period [1/1/1985, 12/31/1990]. On the other hand, the next query is non-temporal, and

displays the current (at time instant t_{now}) *Address* and *PhoneNumber* of *'John Smith':*

Q3: GET < Address, PhoneNumber > of PERSON
WHERE Name = 'John Smith'

As seen from query Q2, the temporal projection of selected entities is specified by a temporal element at the end of the *GET-clause.* The temporal element may be a time period (as in Q2) or may itself be derived from the database for each entity (as in Q4 below). For example, suppose we want the full history of the *Address* and *PhoneNumber* of *'John Smith':*

Q4: GET < Address, PhoneNumber > of PERSON : ET
WHERE Name = 'John Smith'

This retrieves the values of address and phone number over the whole existence time *(ET)* of the entity. If *:ET is* left out, only the current *Address* and *PhoneNumber* (at time instant t_{now}) are retrieved.

Temporal attributes of a role type are *public* to the owner entity type; that is, an owner entity can refer to these attributes (through inheritance) as though they are attributes of the owner entity. Similarly, (temporal and non-temporal) attributes of an entity type are *public* to all associated role types. The definition of attributes and relationships of a role type and its owner entity type as public to each other gives queries the flexibility to specify selection conditions and to retrieve information involving attributes of a role or an entity type by referring to each other's attributes. For example, in queries *Q2, Q3* and Q4, the entity *PERSON* is able to refer to the attributes *Address* and *PhoneNumber* of the entity role *LIVING-PERSON* since the owner of *LIVING-PERSON is* the entity *PERSON.* We can specify similar queries to the queries *Q2, Q3* and Q4 by referring to *LIVING-PERSON* explicitly, as in *Q5, Q6* and Q7, since they only display temporal attributes:

Q5: GET < Address, PhoneNumber > of LIVING-PERSON
: [1/1/1985, 12/31/1990]
WHERE Name = 'John Smith'

Q6: GET < Address, PhoneNumber > of LIVING-PERSON
WHERE Name = 'John Smith'

Q7: GET < Address, PhoneNumber > of LIVING-PERSON : LS
WHERE Name = 'John Smith'

However, *Q6* and Q7will only retrieve entities that are *LIVING-PERSONs* at time t_{now} whereas *Q3* and Q4 may retrieve deceased persons (since conceptual entities have no end time) but then find that their attributes may be *NULL* at time t_{now}.
The projection of (temporal) attributes over a lifespan displays information about

a conceptual entity during the time period it participates as a particular entity role.
For example, in the next query, the history of the *Address* and *PhoneNumber* of *'John Smith'is* retrieved, during the time he was an enrolled student:

Q8: GET < Address, PhoneNumber > of ENROLLED-STUDENT : LS
WHERE Name = 'John Smith'

Here, the *Address* and *PhoneNumber* history are retrieved only during the lifespan *(LS)* that *'John Smith ' exists* in the *ENROLLED-STUDENT* entity role. If *:LS is* left out, the current *Address* and *PhoneNumber* are retrieved if end time *ET(LS)* > t_{now}; if *ET(LS)* < t_{now}, the entity will not be selected since it is not valid as an *ENROLLEDSTUDENT* any more.

The next query retrieves all sections that *'John Smith'has* completed:
Q9: GET < CName of sec-course, Semester, Year > of SECTION
WHERE Name of students of SECTION ⊇ D {'John Smith'}

In this query, there is no need to project the query result over a time period since the attributes *Semester* and *Year,* and the relationship *CS* (specified by *sec-course)* are non-temporal attributes and relationship of *SECTION,* and hence always exist. It is this type of query that becomes cumbersome to specify when no distinction is made between temporal and conceptual objects.. For instance, if the root entity *SECTION* of query *Q9* is replaced by *ACTIVE-SECTION,* we get all sections that *'John Smith'is* currently enrolled in:
Q10: GET < CName of sec-course, Semester, Year > of ACTIVE-SECTION
WHERE Name of e-students of ACTIVE-SECTION ⊇ D {'John Smith'}

This query is implicitly temporal since it refers to the temporal entity role *ACTIVE-SECTION.* The query displays the current (at time instant t_{now}) sections that *'John Smith'is* enrolled in. The capability to express such temporal queries by referring to an entity role without explicit reference to time is one of the advantages of this model.
A temporal query may explicitly specify a temporal projection that is derived from a temporal boolean expression. For example, suppose we want the history of *Name, Office* and *Salary of* each *CURRENT-FACULTY entity* role only when the value of its attribute *Rank* was either *'Assistant Prof'* or *'Associate Prof ':*

Ql1: GET < Name, Office, Salary > of CURRENT-FACULTY
: [(Rank = 'Assistant Prof') OR (Rank = 'Associate Prof)]]

In this case, a different time projection is applied to each selected entity role based upon the time that entity was an assistant or associate professor; that is, the time restriction is correlated to each individual entity role.
When we deal with temporal intervals and elements in *STEER,* we need additional

functionalities that are not needed in other temporal query languages. For instance,*[entity : ET] - [role : LS]* returns the time period (temporal element) when an entity does not participate in a specific role. Hence, to retrieve the *Name, SSN, a*nd *Salary of* each faculty during the time period she/he is not *CURRENT-FACULTY* (e.g. on sabbatical or working for industry), we write:

Q12: GET < Name, SSN, Salary > of FACULTY
: [FACULTY : ET] - [CURRENT-FACULTY : LS]

Here, the *Name, SSN,* and *Salary* of a faculty are retrieved only during the period *[FACULTY : ET] - [CURRENT-FACULTY : LS],* which is different for each selected entity. Note the difference between the temporal expression in queries *Q11* and *Q12.* In both queries *Qll* and *Q12,* temporal element constructor operators are used to define temporal elements at the end of the *GET-clause.* However, in query *Q11,* the boolean condition c = (((Rank = *'Assistant Prof') OR (Rank =- 'Associate Prof'))* is based on a boolean predicate that involves attribute values of an entity role, whereas in query *Q12,* the boolean condition refers only to the existence time of *FACULTY* and the lifespan of *CURRENT-FACULTY.* In query *Q11,* the temporal element at the end of the *GET-clause* is the true_time of a boolean condition, whereas in query *Q12,* the temporal element is the difference between two true_times, namely the existence time of a *FACULTY entity* and its lifespan as a *CURRENT-FACULTY* entity role.
The next query retrieves the history of the *Name, Address* and *PhoneNumber* of living persons during the period they were not enrolled students:

Q13: GET < Name, Address, PhoneNumber > of PERSON
: [LIVING-PERSON : LS] - [ENROLLED-STUDENT : LS]

The usual set theoretic operations of *UNION, INTERSECTION, DIFFERENCE* and *COMPLEMENT* can be combined with temporal element constructor operators. Both previous queries *Q12* and *Q13* use the *DIFFERENCE* operator. The next query uses the *COMPLEMENT* operator to retrieve the history of the *Name, Address* and *PhoneNumber* of persons before they become faculty members:

Q14: GET < Name, Address, PhoneNumber > of PERSON
: COMPLEMENT [[FACULTY : ET]

The idea of applying a temporal boolean condition to entity roles and entities can be extended to temporal attributes. The true_time of a boolean condition reduced to a temporal attribute name is represented as [[*temporal_attribute : time_period]".* This corresponds to the true_time of the *temporal_attribute* during *time_period.* For example,
the next query retrieves the history of the *Name, StudentNo, CName, Semester* and *Year* of enrolled students during the period they had a valid *Classification* (that is, a *Classification* value that is not *NULL):*

Q15: GET < Name, StudentNo, < CNazne of sec-course, Semester, Year > of sections > of ENROLLED-STUDENT : [Classification : LS]

- **Temporal Selection**

Next, we consider the specification of temporal conditions to select entities. These will usually involve the specification of temporal selection predicates in the *WHERE-clause.* For example, consider the query to retrieve the *Name* and *PhoneNumberof all* persons who lived in *'Arlington' on* 3/30/1992:

Q16: GET < Name, PhoneNumber > of LIVING-PERSON : [3/30/1992]
WHERE [Address = 'Arlington'] ⊇ [3/30/1992]

In query *Q16,* the *WHERE-clause* is a temporal selection condition. For each *LIVING-PERSON entity* role, it first calculates the temporal boolean expression c = *(Address = 'Arlington');* if the true_time [c] ⊇ [3/30/1992], the temporal selection condition evaluates to *YES* and the *LIVING-PERSON entity* role is selected by the *WHERE-clause. However,* it is still necessary to specify the temporal projection [3/30/1992] again in the *GET-clause* since leaving it out would retrieve the current *Name* and *PhoneNumber* of each selected entity rather than those on 3/30/1992.
The next query retrieves the *SectionNumber* and *ClassRoom* of all active sections that were held in room *'EB119'during* the period 1990-1991:

Q17: GET < SectionNumber, ClassRoom > of ACTIVE-SECTION
WHERE ([ClassRoom = 'EB119'] ∩ [1/1/1990, 1/12/1991]) ≠ 0

When we deal with time periods, we sometimes need to access the first and last time points of temporal elements. For example, to retrieve the *Name*, *SSN* and *Address* of all current students who lived in *'Arlington'* when they first enrolled as a student, we write:

Q18: GET < Name, SSN, Address > of ENROLLED-STUDENT
WHERE [Address = 'Arlington'] ⊇ ST(LS)

Here, the temporal selection condition evaluates to *TRUE if [c] ⊇ ST(LS),* where *c = (Address = 'Arlington').* The term *ST(LS)* means the start time point of a lifespan. Note that *ST(LS)* is implicitly applied to *ENROLLED-STUDENT* since it is the root entity role. This can also be written as *ST([ENROLLED-STUDENT : LS]*)
The lifespan of an entity role can be a continuous time period. This may happen if either an entity role has come into existence in the mini-world and never ceased to exist, or an entity role has come into existence for a while then has ceased to exist and has never re-existed in the mini-world. In order to support the concept of continuous and discontinued lifespans in our query language, we introduce the keywords *CONTINUOUS* and

DISCONTINUED. For example, suppose we want to display the courses that have been continuously taught every semester:

Q19: GET < Cname, CNumber, Dept > of VALID-COURSE
WHERE CONTINUOUS LS

This is similar to the temporal *ALWAYS SINCE* operator in temporal logic. As a final example, note that a name related with any lifespan besides the root entity mast be explicitly specified in a temporal query. For instance, the next query explicitly specifies the lifespan of attribute *Address* in the *WHERE-clause ,* and retrieves the *Name, SSN* and *Address* of all current students whose initial *Address* value was *'Arlington':*

Q20: GET < Name, SSN, Address > of ENROLLED-STUDENT
WHERE [Address = 'Arlington'] ⊇ ST([Address: LS])

- **Temporal Version Restriction Operators**

In the *STEER* data model, the complete history of an entity (or an entity role) is kept. The temporal versions of an entity (or an entity role) are ordered and queries may be restricted to specific versions of an entity (or an entity rote). A temporal version restriction operator may be specified in the *GET or WHERE* clause of temporal *GORDAS* queries. The syntax of our version restriction operator is:

: ([NAME] : INTERVAL <INDEX>)

where the term [*NAME*]: is optional and the term *INTERVAL* < *INDEX* >is required. The term [*NAME]* is a true_time, where *NAME* may be either a boolean condition, or may be reduced to an entity name, an entity role name, or a temporal attribute. The term *INTERVAL* < *INDEX* > indicates a projection either over a single interval if < *INDEX* > is an integer or over a range of intervals if < *INDEX* > is an integer range. (Note that we assume that the intervals of a temporal element are disjoint and in the canonical temporal element representation.)

As an example, the version restriction operator :(*INTERVAL* 1), when applied to *a CURRENT-FACULTY entity* role *ro* (Figure1) restricts the temporal element to the first interval of its lifespan. In this case, the term [*NAME*]: is not used in the version restriction operator :(*INTERVAL 1).* However, if the term [*NAME]*: is used in the version restriction operator such as *:([Address]~ : INTERVAL 1),* then when it is applied to a *CURRENT-FACULTY* entity role *ro* (Figure 1) it restricts the temporal element to the first interval of the lifespan of attribute *Address.*

The next query retrieves the *Name* and the first three *Salary* values for each faculty:

Q21: GET < Name, Salary : (INTERVAL 1 to 3) > of FACULTY

The term *:(INTERVAL 1 to 3)* in the *GET-clause* specifies that the projection displays the first three *Salary* values for each *FACULTY.* Notice that once a temporal version

restriction operator appears in either the *GET* or *WHERE* clause of a query, we immediately deal with the full temporal entity in that clause, rather than the current entity version only.

Temporal operators may be nested and are evaluated from *left to right.* For example, suppose we want to display the *Name, SSN* and the current *Address* for each person whose first *Address* was *'Houston'* and third *Address* was *'Arlington':*

Q22: GET < Name, SSN, Address > of PERSON
WHERE (Address: (INTERVAL 1) = 'Houston') AND
(Address: (INTERVAL 3) = 'Arlington')

The term *Address : (INTERVAL 1) = 'Houston' in* the *WHERE-clause* means that we first apply the temporal ordering restriction operator *:(INTERVAL 1)* and then compare it with *= 'Houston'*. Similarly, the term *Address : (INTERVAL 3) = 'Arlington' in* the *WHERE-clause* means that we first apply the temporal ordering restriction operator *:(INTERVAL 3)* and then compare it with *= 'Arlington'*.

As seen from queries Q21 and Q22, if the term [*NAME*]: is omitted from the version restriction operators, then the term *INTERVAL < 1NDEX >* is applied to the specific attribute. However, if we would like to display the *Name* and *PhoneNumber* of a person during the time period she/he first lived in *'Arlington'*, we could write:

Q23: GET < Name, PhoneNumber : ([Address = 'Arlington']
: INTERVAL 1) > of PERSON

In this case, the true_time of the boolean expression *c = (Address = 'Arlington'*) is evaluated for each entity and then the temporal element is assigned to the first interval of each true_time. Note that the projection over *PhoneNumber* may result with multiple values. However, we could even further restrict the previous query, *Q23,* by displaying only the first value of the *PhoneNumber:*

Q24: GET < Name, PhoneNumber : ([Address='Arlington'
: INTERVAL 1) : (INTERVAL 1) > of PERSON

Temporal version restriction operators are not limited to attributes; they may be applied to entities and therefore restrict queries to a specific range of lifespans. For example, the next query displays the *Name, SSN, Address, PhoneNumber, CName, Semester, Year* during the second interval of the lifespan of each *ENROLLEDSTUDENT* who currently lives in *'Arlington':*

Q25: GET < Name, SSN, Address, PhoneNumber, < CName of course,
Semester, Year > of a-sections > of ENROLLED-STUDENT
: (INTERVAL 2)
WHERE Address = 'Arlington'

As a final example, note that any restriction condition specified on an entity is applied before any other restriction operator is applied to its attributes. Hence, if we would like to display for current full professors, their *Name*, and the initial *Salary* as associate professors, we could write:

Q26: GET < Name, Salary : (INTERVAL 1) > of CURRENT-FACULTY
: [Rank = 'Associate Prof']
WHERE Rank = 'Full Prof'

- **Temporal Scope Operators**

In the *GORDAS* language, one can reference the attributes of an entity related to the root entity by using a connection name. In the temporal *GORDAS,* related entities must be projected over the temporal elements of connection names. To generalize our temporal projection capabilities, we introduce the scope operator, denoted by *SCOPE,* which overwrites the temporal projection of a root entity (or related entities). For example, if we would like to retrieve the *Name* and *Rank* attribute values of each current faculty during their *LAST* - 1 interval but we would like to retrieve their initial *Salary,* we could write:

Q27: GET < Name, Rank, Salary : SCOPE(INTERVAL 1) >
of CURRENT-FACULTY : (INTERVAL LAST - 1)

In this case, the *SCOPE* operator at the end of *Salary* attribute overwrites the temporal projection at the end of the *GET-clause.*