**Entity Relationship Diagram (ERD)**

Identify the 1. Entities; 2. Relationships; 3. Attributes, Participations, Cardinalities

**Entity Set:** The current state of the entities of the entity type in the DB

**Attributes:** Data type associated with it sometimes called a value set

**Types of Attributes:**

* Key Attribute (underlined, require unique value, can have multiple keys)
* Composite Attribute
* Multivalued Attribute (represented as double ellipse)
* Derived Attribute (represented as dotted ellipse)
* Attributes can have multiple types

**Relationship Set:** Current state of a relationship type in the DB

**Constraints on Relationships:**

* Cardinality Ratio (specifies maximum participation):

(1:1), (1:N) or (N:1), (M:N)

* Existence Dependency Constraint (specifies minimum participation)

AKA: Participation Constraint

Optional (0) or mandatory (1+)

* Multiple labelled constraints to the same relationship:   
  Recursive Relationship Type (distinct roles, self-referencing relationship type)

**Participation Constraints:**

* Total Participation (double lines) or Partial Participation (single lines)

**Weak Entity Types**

* Doesn’t have key attribute, identification dependent on another entity type
* Identified by the combination of:

Partial key of the weak entity (dotted line under the label)

The other entity in the identifying relationship (double diamond)

**N-Ary Relationships (N>2)**

* Can represent different information than a single ternary relationship
* Can be represented as a weak entity if the data model allows multiple identifying relationships

**Extended Entity Relationship Diagram (EERD)**

* Includes Super/Subclasses, Specializations/Generalizations, Categories+Unions
* Subclasses inherit all attributes of a superclass
* EERDs can be converted to ERDs by replacing subclass designation with a relationship (typically `IsA`)

**Membership Constraints:**

* Predicate Defined: By some predicate (Job\_Type = “Engineer”)
* Attribute Defined: Subclasses are dependent on the same attribute
* User Defined: No condition specified

**Disjoint Constraints**

* Disjoint: Subclasses are mutually exclusive (circled d)
* Overlapping: Not mutually exclusive (circled o)
* A “U” is drawn on the line connecting the d/o to the subclass (round side points)

**Completeness Constraint**

* Total: Every entity in superclass must be a member of some subclass (=)
* Partial: Allows an entity to not belong to any subclass (single line -)

**Category and Union:**

* A subclass that has multiple super classes (category or UNION type)
* Denoted by a circled U, as with disjoint constraints
* A “U” is drawn on the line connecting the U to the subclass

**Relational Model (RM)**

* Represents the database as a collection of relations (tables of values)
* Schema: Header of the table (attributes)
* Tuple: Ordered collection of values

**Relational Model Constraints:**

* Inherent/Implicit: Based on data model itself
* Schema-based/Explicit: Expressed in the schema (max cardinality ratio)
* Application-based/Semantic: Must be specified and enforced in the programs

**Key Constraints:**

* Superkey of R: Set of attributes where no two tuples will have the same values
* Key of R: Minimal superkey, removal of any attribute makes it no longer a key

**Entity Integrity:**

* Null is not allowed as an attribute of a primary key

**Referential Integrity Constraints:**

* Used to specify relationship among tuples in different relations
* Referencing attribute draws arrow pointing to referenced attribute

**Update Operations on Relations:**

* INSERT, DELETE, MODIFY
* Integrity constraints should not be violated, updates may propagate to cause other updates automatically to maintain constraints
* Several actions can be taken:

Cancel: REJECT/RESTRICT

Perform, notify the user

CASCADE, SET NULL

Execute user-specified error-correction routine

**Equivalent Relational Database Schema:**

* Equivalent if they have the same structure (plus integrity constraints)
* Rename attributes such that two schemas are identical = equivalent

**Conversion of ERD to RM Algorithm**

1. Mapping of regular entity types

Break composite attributes to simple counterparts

2. Mapping of weak entity types

Include PK of owner entities as foreign key

3. Mapping of binary 1:1 relationships

Three approaches:

One relation with total participation adds foreign key to the other relation

Merge both entities and relationship into single relation

New relation, pair of primary keys of both entities

4. Mapping of binary 1:N relationships

Primary key of 1-side becomes foreign key of N-side

5. Mapping of binary M:N relationships

New relation, pair of primary keys of each entity

6. Mapping of multivalued attributes

New relation, primary key is foreign key of entity plus multivalued attribute

7. Mapping of n-ary relationships

New relation, same idea as M:N

8. Mapping specialization and generalization

Multiple relations – superclasses and subclasses (always works)

Multiple relations – subclasses only (only full specializations)

Single relation with 1+ discriminating attributes (always works)

9. Mapping of union types (categories, EERD only)

Add primary key of superclass into subclass/category

Add attributes of subclasses considering them

For mapping category whose defining superclasses have different keys, customary to specify a new key attribute (surrogate key) when creating relation to correspond to the category.

**Relational Algebra**

SELECT (σ): Selects a subset of tuples from a relation based on a condition

PROJECT (π): Results in a relation with only the requested attributes

Removes duplicates

RENAME (ρ): ρ S(B1, B2, ..., BN) (R): Renames R to S, and R’s attributes to (B1, B2, …, BN)

UNION (∪): Stick two tables together, duplicates are eliminated (type compatible)

INTERSECTION (∩): Find common elements (type compatible)

DIFFERENCE (-): R – S, All tuples in R that are not in S (type compatible)

CARTESIAN PRODUCT (x): Multiple each tuple in R1 with each tuple in R2

JOIN (⨝): Cartesian product with a join

EQUIJOIN: Join with equality comparisons only (=)

NATURAL JOIN (\*): Joins relations based on attributes with the same name

DIVISION (÷): See example

**Query Tree Notation:**

* Tree structure to represent a query, nodes stand for operations and leaves represent base relations

**Aggregate Function Operation (Ƒ):**

* MAX, MIN, AVG, COUNT
* Grouping attribute placed to the left of Ƒ, aggregate functions to the right

**Outer Joins**

* Populate all of the tuples of the side listed, values on the other side that do not satisfy the condition are populated with null.

**Tuple Relational Calculus (RC)**

* No order of operations, specifies only what info the result should contain
* Reminder: p -> q is equivalent to (not p) or q

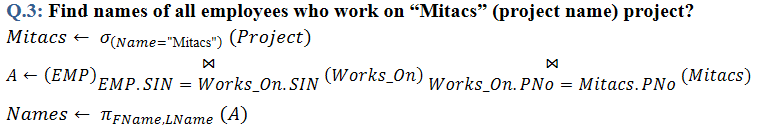
**Quantifier Equivalence:**

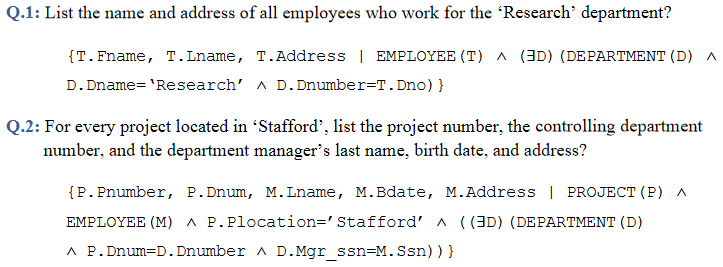
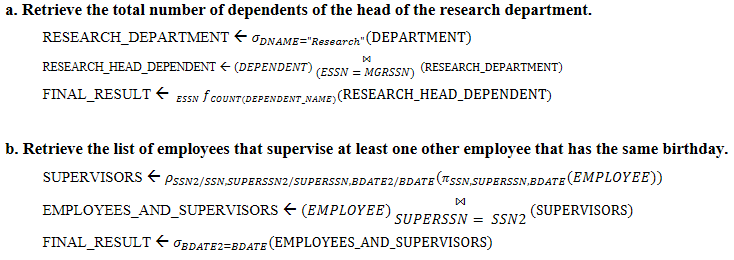
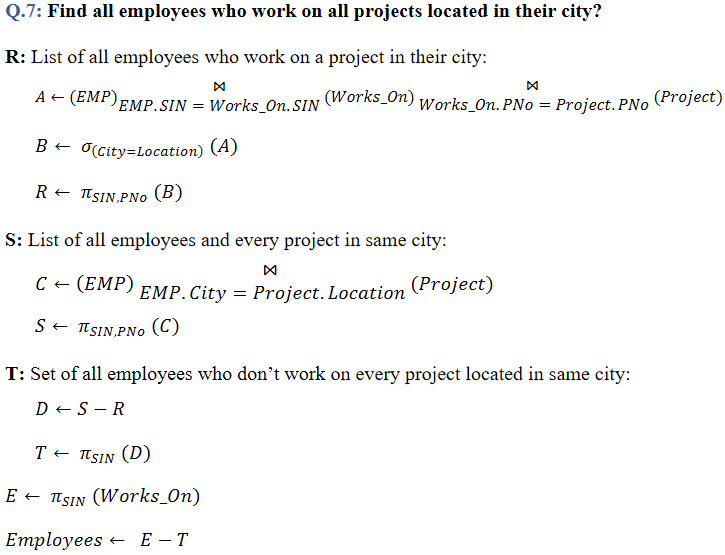
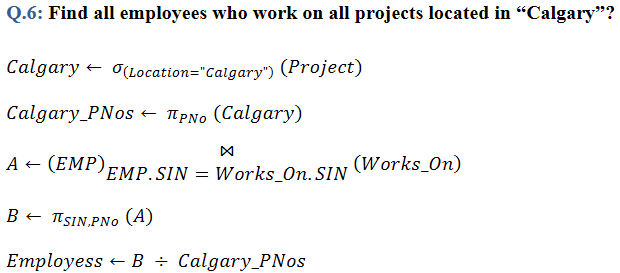
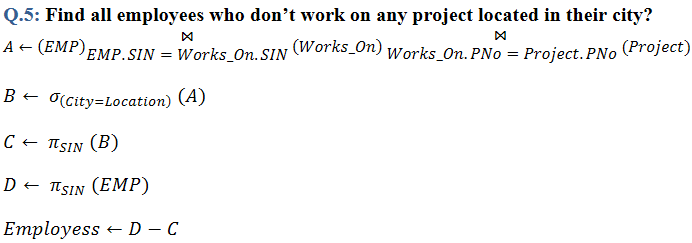
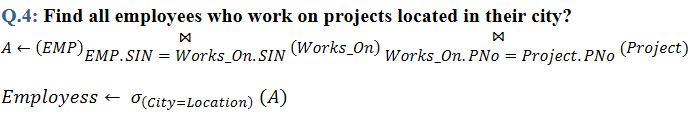
* ∀x P(x) is equivalent to not ∃x (not P(x))
* ∃x P(x) is equivalent to not ∀x (not P(x))

**DeMorgan’s Rules:**

* not (p and q) = (not p) or (not q)
* not (p or q) = (not p) and (not q)

**RA Examples:**





**SQL**

**Basic Attribute Data Types:**

* INTEGER/INT, SMALLINT
* FLOAT/REAL, DOUBLE PRECISION
* CHARACTER(n) / CHAR(n) (fixed length)
* CHARACTER VARYING (n) / VARCHAR(n) / CHAR VARYING(n)
* BIT(n), BIT VARYING(n)
* TRUE, FALSE, NULL
* DATE (YYYY-MM-DD)

**Domains:**

* Example: CREATE DOMAIN SSN\_TYPE AS CHAR(9);
* CREATE TABLE EMP (SSN SSN\_TYPE NOT NULL, … );

**Basic Constraints:**

* Key Constraint: primary key value cannot be duplicated
* Entity Integrity Constraint: Primary key value cannot be null
* Referential Integrity Constraints: Foreign key must have a value that is already present as a primary key, or may be null
* NOT NULL, CHECK (condition), PRIMARY KEY, UNIQUE, FOREIGN KEY

**Specifying Constraints on Tuples using CHECK**

* CHECK clauses at the end of CREATE TABLE, apply to each tuple individually
* CHECK (DeptCreateDate <= MgrStartDate);

**DISTINCT**:

* Use to eliminate duplicate tuples from query results
* SELECT DISTINCT Salary FROM EMPLOYEES;

**Set Operations**

* UNION, EXCEPT, INTERSECT
* Type compatibility is needed for these operations to be valid

**LIKE and BETWEEN**

* LIKE used for string pattern matching. % replaces arbitrary number of 0+ chars, \_ replaces a single character (WHERE Address LIKE ‘%Houston,TX%’;)
* BETWEEN comparison operator (WHERE (Salary BETWEEN 3 AND 4) AND … ;)

**Ordering of Query Results:**

* ORDER BY D.Dname DESC, E.Lname ASC, E.Fname ASC
* Typically at the end of the query

**INSERT**

* INSERT INTO EMPLOYEE VALUES (…);

**Bulk Loading of Tables**

* Creates more than one tuple in the DB, needs to extract data from the DB
* INSERT INTO table SELECT … ;
* CREATE TABLE table LIKE other\_table;

**DELETE:**

* DELETE FROM table WHERE condition;

**UPDATE:**

* UPDATE table SET attribute=value WHERE condition;

**NULL:**

* NULL = NULL is avoided, typically evaluates to false
* IS NULL, IS NOT NULL

**ANY (or SOME):**

* SELECT … FROM table WHERE attribute > ANY (nested select);
* Operators to be combined with ANY: =, >, >=, <, <=, <>

**UNIQUE and EXISTS:**

* Boolean functions, check for duplicates and check if nested query is not empty
* No direct implementation of ∀ in SQL, quantifier equivalence shown on page 2

**JOIN**

* SELECT … FROM A JOIN/INNER JOIN B ON conditions;
* SELECT … FROM (A NATURAL JOIN B AS (…));
* LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN

**Aggregate Functions:**

* SELECT SUM(Salary), MIN(Salary), MAX(Salary), AVG(Salary) FROM … ;
* SELECT COUNT(\*) FROM …;

**GROUP BY:**

* Partition relation into subsets of tuples by a grouping attribute
* SELECT DNO, COUNT(\*), AVG(Salary) FROM EMPLOYEE GROUP BY DNO;
* SELECT Pnumber, Pname, COUNT(\*) FROM … GROUP BY Pnumber, Pname;
* Grouping is applied to the result of the join

**HAVING:**

* Provides a condition to select or reject an entire group
* WHERE is applied first, then HAVING
* SELECT … WHERE condition GROUP BY … HAVING COUNT(\*) > 2;

**WITH:**

* Define a temporary “View” table for a particular query
* WITH BIGDEPTS(DNO) AS (SELECT DNO FROM EMPLOYEE GROUP BY DNO HAVING COUNT(\*) > 5)

SELECT DNO, COUNT(\*) FROM EMPLOYEE WHERE … AND DNO IN BIGDEPTS GROUP BY DNO;

**CASE:**

* UPDATE EMPLOYEE SET SALARY =

CASE WHEN DNO = 1 THEN SALARY + 2000

CASE WHEN DNO = 4 THEN SALARY + 1000

ELSE SALARY + 0;

**ASSERTION and TRIGGER:**

* Specify additional constraints and automatic actions (examples below)

**VIEWS (Virtual Tables) in SQL:**

* Single virtual table derived from other defining tables
* CREATE VIEW View2(Att1, Att2, Att3) AS SELECT A, B, C FROM TableA, TableB;
* Views are always up to date (DBMS), DROP VIEW to dispose of it
* Once the view is defined, SQL queries can use it in the FROM clause
* Cannot update views that contain aggregate functions
* Can be used to hide certain attributes/tuples from unauthorized users

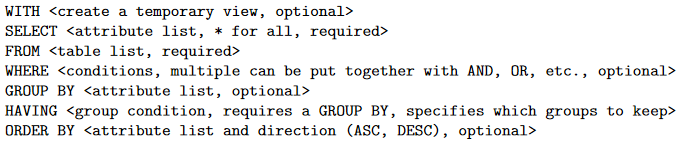
**DROP:**

* Behaviour options: CASCADE, RESTRICT
* DROP TABLE name RESTRICT; drop if table has no constraints or routines
* DROP TABLE name CASCADE; drop table with associated constraints or routines

**ALTER:**

* ALTER TABLE name ADD COLUMN a VARCHAR(12);
* ALTER TABLE name DROP COLUMN b CASCADE;
* ALTER TABLE name MODIFY c INT; (change type)

**General Query Order:**

****

**Normal Forms**

**Armstrong Inference Rules:**

IR1. Reflexive: If Y ⊆ X, then X -> Y

IR2. Augmentation: If X -> Y then XZ -> YZ

IR3. Transitive: If X -> Y and Y -> Z then X -> Z

IR1, IR2, IR3 form a sound and complete set of inference rules

* These rules hold and all other rules that hold can be deduced from these

IR4. Decomposition: If X -> YZ, then X -> Y and X -> Z

IR5. Union: If X -> Y and X -> Z then X -> YZ

IR6. Psuedo-Transitive: If X -> Y and WY -> Z then WX -> Z

* All can be deduced from IR1, IR2, IR3 (completeness)

**Attribute Closure:**

* To compute F+, find all attributes X that appear on the left hand side of each FD in F, then find each attribute that functionally depends on X (X+)

X+ is the closure of X under F

* **Definition:** Given a set of FDs F, for a set of attributes X, X+ is the set of attributes that are functionally determined by X based on F
* **Algorithm to determine closure:**
* Input: A set F of FDs on a relation schema R, a set of attributes X, X ⊆ R

X+ <- X

repeat

for each Y -> Z in F do

if Y ⊆ X+ then X+ <- X+ ∪ Z

**Minimal Covers of FDs:**

* Reduce the set F to its minimal form such that the minimal set is still equivalent to the original set of F
* An attribute is extraneous if we can remove it without changing the closure of the set of dependencies

**Minimal Sets of FDs Conditions:**

1. Every dependency in F has a single attribute for its RHS (canonical form)
2. No redundant dependencies in F (no FDs can be inferred from the rest of F)
3. No extraneous attributes in LHS

* **Minimal Cover** is a minimal set of dependencies that is equivalent to F, there will always be at least one.
* Convert F to canonical form, remove extraneous attributes and redundant FDs

**Calculating a Key for a Relation:**

* Superkey of R: A set of attributes SK of R where no two tuples in any valid relation state will have the same value for SK
* Key of R: A minimal superkey
* Multiple keys are candidate keys, one is arbitrarily selected as the primary key and the others are called secondary keys.
* Prime attributes are attributes of any candidate key (else nonprime)

**Transitive FD:**

* An FD X -> Y is transitive if:

There exists a set of attributes Z in R that is neither a candidate key nor a subset of any key in R; and both X->Z and Z->Y hold

Basically, nonprime that defines a nonprime

**Normal Forms Based on Keys:**

* 1st normal form (1NF)

- All attributes depend on the key

* 2nd normal form (2NF)

- All attributes depend on the whole key

* 3rd normal form (3NF)

- All attributes depend on nothing but the key

* BCNF

- All key attributes must define every other attribute

- For every non-trivial FD for R, X -> A, X is a superkey for R

**First Normal Form (1NF):**

* Disallows composite attributes, multivalued attributed, nested relations
* Tend to end up with redundancy

**Second Normal Form (2NF):**

* Every non-prime attribute in R is fully functionally dependent on every key of R

**Third Normal Form (3NF):**

* Every non-prime attribute in R is non-transitively dependent on every key of R
* Alternative: Whenever FD X -> A holds in R then either:

X is a superkey of R, or A is a prime attribute of R

**Boyce-Codd Normal Form (BCNF):**

* Whenever an FD X -> A holds in R, then X is a superkey of R

**Lossless Join (aka Additive Join):**

* The addition of spurious information (loss of information)
* Example and simplified algorithm below

**Testing Non-Additivity of Binary Relational Decompositions**

* Binary Decomposition: Decomposing R into two relations
* A decomposition D = {R1, R2} has the lossless join property iff either of the following FDs is in F+:

R1∩R2 -> R1 – R2

R1∩R2 -> R2 – R1

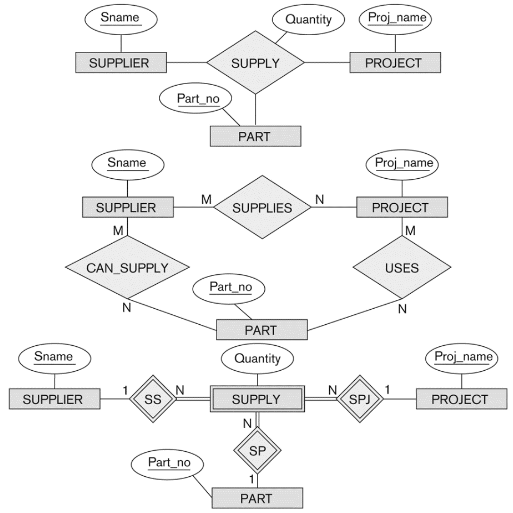
**Successive Non-Additive Join Decomposition**

* If D= {R1, …, Rm} of R has the lossless join property
* And if D1 = {Q1, …, Qk} of Ri has the lossless join peroperty
* Then the decomposition D’ = {R1, …, Ri-1, Q1, …, Qk, Ri+1, …, Rm} also has the lossless join property.

**Examples:**

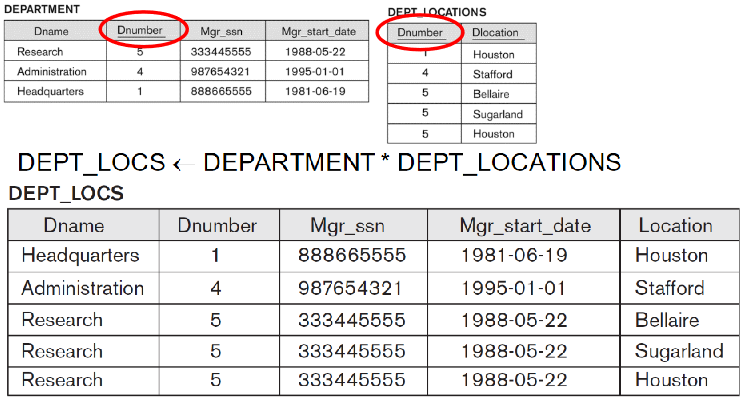
**Ternary Relationships: (ERD)**

One ternary relationship, two ways to otherwise describe it

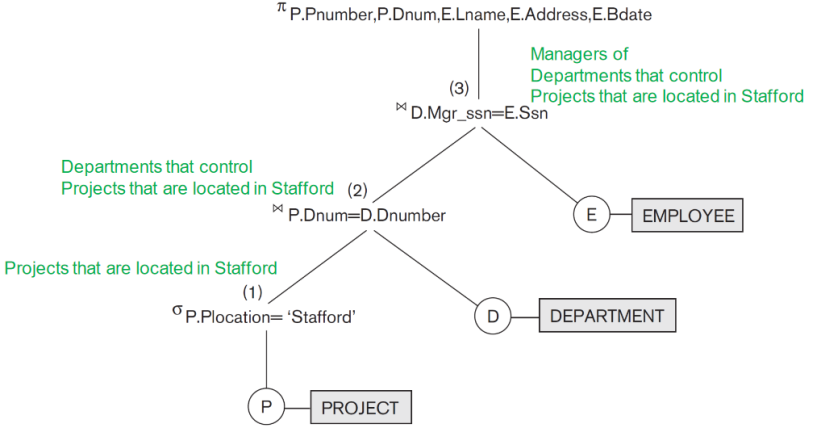


|  |  |
| --- | --- |
| **Cartesian Product (RA)** | **DIVISION (RA)** |

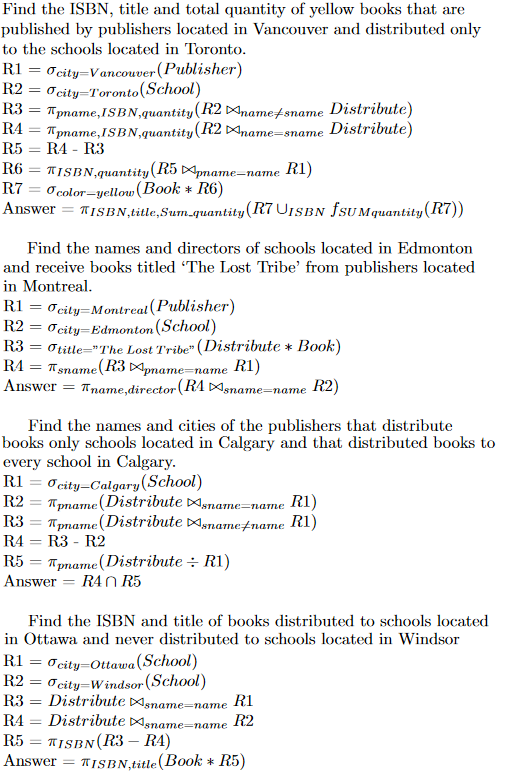
**Natural Join (RA)**

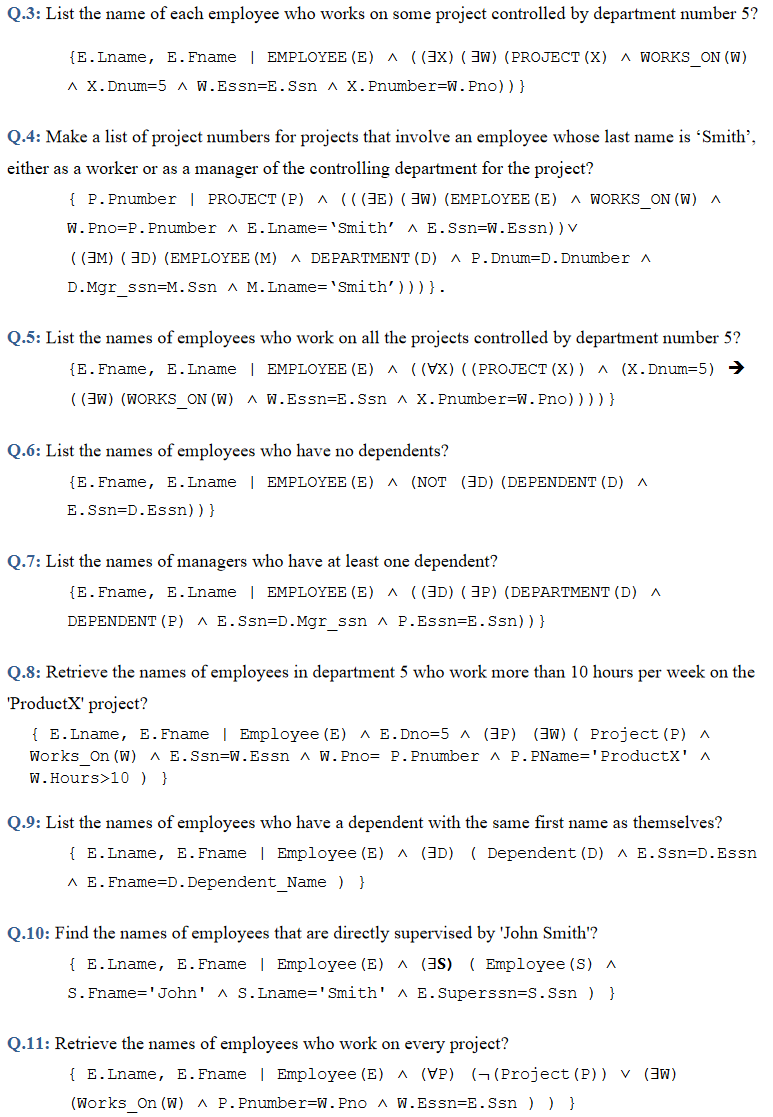
****

**Query Tree Notation (RA)**

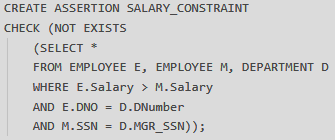
****

**Assignment 3 (RA)**

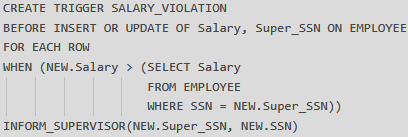
****

**TRC Tutorial Examples (TRC)  
**

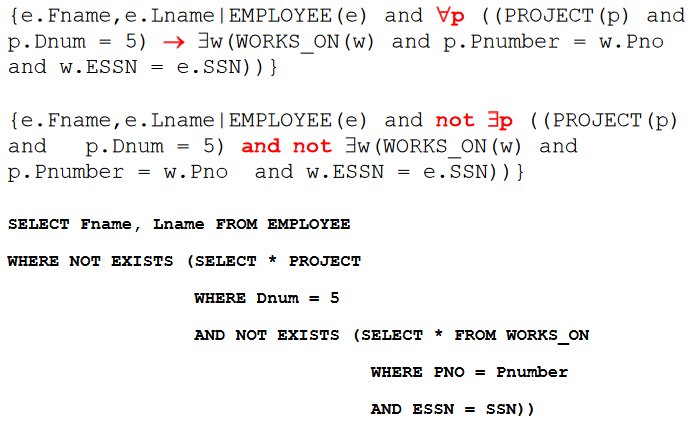
**CREATE ASSERTION (SQL)**



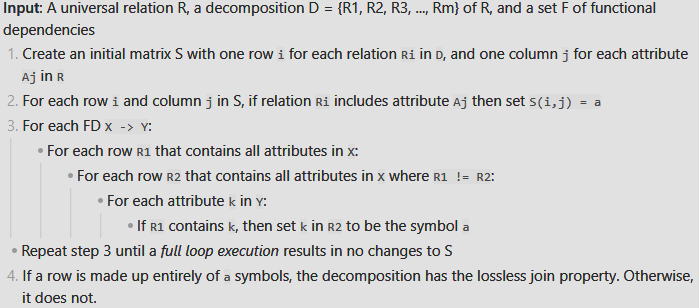
**CREATE TRIGGER (SQL)**

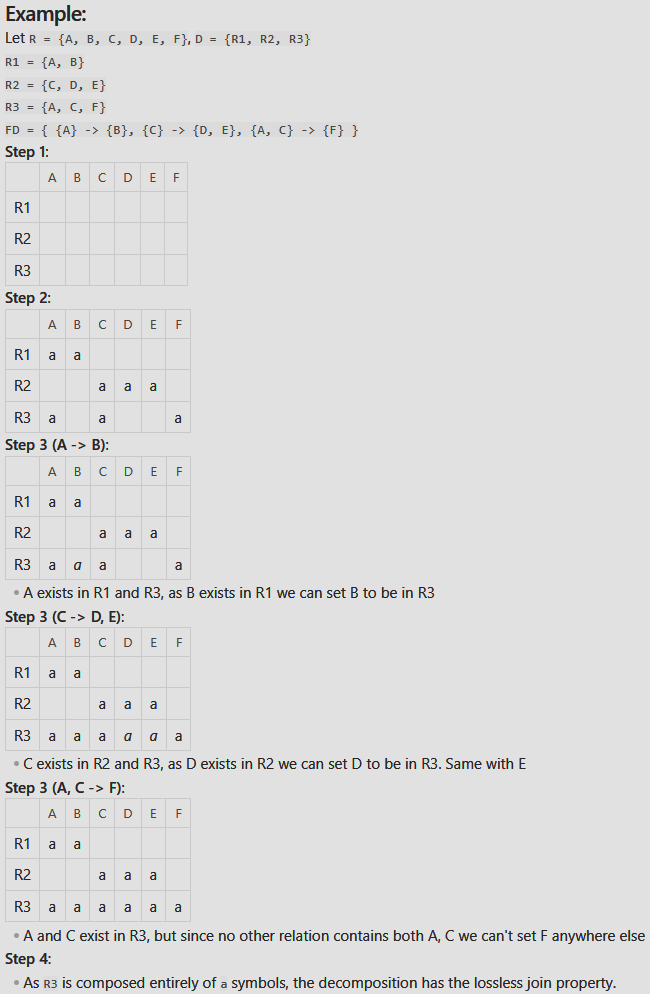
****

**Converting TRC to SQL (SQL)**

****

**Lossless Join (NF)**





**SQL Injection (SQL)**

