MIS 331
Database Management Systems

Lecture Notes
(Student Version)

Instructor

Jinsoo Park

Department of Management Information Systems
College of Business and Public Administration
The University of Arizona
Tucson, AZ 85721
E-mail: jpark@bpaosf.bpa.arizona.edu
Web Page: http://jpark.bpa.arizona.edu

1 9 9 8

Table of Contents

Relational Algebra

- Procedural language
- Six basic operators
  - SELECT
  - PROJECT (& Generalized Projection)
  - UNION
  - Set DIFFERENCE
  - Cartesian Product
  - RENAME
- Additional operations that do not add any power to the relational algebra, but that simplify common queries.
  - Set INTERSECTION
  - NATURAL JOIN
  - DIVISION
  - ASSIGNMENT
- Extended relational algebra operations
  - OUTER JOIN
  - Aggregate Functions
- Modification of the Database
- View
- More examples (exercises) for relational algebra

Relational Algebra

- Relational algebra is a *procedural* language in which a user specifies *what* data is needed and *how* to get it.
  - We must write a *sequence of operations*. A certain order among the operation is always explicitly specified.
  - The order also specifies a partial strategy for evaluating the query.
  - *Relational calculus* — A declarative, *non-procedural* language (thus, the relational calculus is often considered to be a higher-level language than the relational algebra). There is no description of how to evaluate a query; a calculus expression specifies *what* to be retrieved rather than how to retrieve it.
  - Note that both relational algebra and relational calculus are identical; that is, any retrieval that can be specified in the relational algebra can be specified in the relational calculus, and vice versa. In other words, the *expressive power* of the two languages is *identical*.

- It consists of a set of operations that take one or two relations as input and produce a new relation a their result.

- The result of each operation is a *relation*, which can be further manipulated, assigned to a temporary relation, or assigned to an existing relation (which results in the modification of an existing relation).

SELECT Operation

- Notation: $\sigma_P(r)$

- Defined as:

$$\sigma_P(r) = \{t \mid t \in r \textbf{ and } P(t)\}$$

  Where *P* is "selection condition(s)"; a formula in propositional calculus, dealing with terms of the form:

<attribute> = <attribute> or <constant>
≠
>
≥
<
≤
"connected by": $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)

- To select a *subset* of the tuples in a relation that satisfy a *selection condition*, *P*.

- The relation resulting from the SELECT operation has the *same attributes* as the relation specified in relation name, (*r*).

SELECT Operation (Cont.)

- *Unary*
  - The operator is applied on a *single relation* (cannot be used to select tuples from more than one relation).
  - The operator is applied to *each tuple individually*; hence selection conditions cannot apply over more than one tuple.

- The *degree* of the resulting relation is the same as that of the original relation R on which the operation is applied, because it has the same attributes as R.

- The number of tuples in the resulting relation is always *less than or equal to* the number of tuples in the original relation R. The fraction of tuples selected by a selection condition is referred to as the *selectivity* of the condition.

- *Commutative*

  $\sigma_{\text{<cond1>}}(\sigma_{\text{<cond2>}}(r)) = \sigma_{\text{<cond2>}}(\sigma_{\text{<cond1>}}(R))$

  Hence, a sequence of SELECTs can be applied in any order.

- We can always combine a *cascade* of SELECT operations into a single SELECT operation with a conjunctive (**AND**) condition, $\wedge$.

  $\sigma_{\text{<cond1>}}(\sigma_{\text{<cond2>}}(\ldots(\sigma_{\text{<cond}n\text{>}}(R))\ldots)) = \sigma_{\text{<cond1>} \wedge \text{<cond2>} \wedge \ldots \wedge \text{<cond}n\text{>}}(R)$

SELECT Operation - Example

- Relation r:

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- $\sigma_{A = B \wedge D > 5}\ (r)$

A	B	C	D
α	α	1	7
β	β	23	10

PROJECT Operation

- Notation:

$$\Pi_{A_1,\, A_2,\, \ldots,\, A_k}(r)$$

  where A_1, A_2 are attribute names and r is a relation name.

- The result is defined as the relation of k columns obtained by erasing the columns that are not listed.

- The resulting relation has only the attributes specified in <attribute list> and *in the same order as they appear in the list*. Hence, its *degree* is equal to the number of attributes in <attribute list>.

- *Duplicate elimination* — The operation implicitly *removes any duplicate tuples*, since relations are sets. This is necessary to ensure that the result of the PROJECT operation is also a relation—a set of tuples.

- The number of tuples in a resulting relation is always *less than or equal to* the number of tuples in the original relation.

- *No commutativity*

  $\pi_{<list1>}(\pi_{<list2>}(R)) = \pi_{<list1>}(R)$ as long as <list2> contains the attributes in <list1>; otherwise, the left-hand side is incorrect. Thus, the commutativity does not hold on PROJECT.

PROJECT Operation - Example

- Relation r:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

- $\Pi_{A,\ C}\ (r)$

A	C
α	1
$\cancel{\alpha}$	$\cancel{1}$
β	1
β	2

=

A	C
α	1
β	1
β	2

Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the projection list.

$$\Pi_{F_1,\ F_2,\ \ldots,\ F_n}(E)$$

- E is any relational-algebra expression

- Each of F_1, F_2, …, F_n are arithmetic expressions involving constants and attributes in the schema of E.

- Given relation CREDIT-INFO(Customer-name, Limit, Credit-balance), find how much more each person can spend:

$$\Pi_{\textit{Customer-name, Limit} - \textit{Credit-balance}}\ (\text{CREDIT-INFO})$$

UNION Operation

- Notation: $r \cup s$

- Defined as:

$$r \cup s = \{t \mid t \in r \textbf{ or } t \in s\}$$

- For $r \cup s$ to be valid (*Union compatibility*),
  - r, s must have the *same number of attributes*.
  - The attribute domains must be compatible (e.g., 2nd column of r deals with the same type of values as does the 2nd column of s).

- It is a binary operation.

- *Commutative*

$$r \cup s = s \cup r$$

- *Associative*

$$r \cup (s \cup t) = (r \cup s) \cup t$$

UNION Operation - Example

- Relation *r, s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cup s$

A	B
α	1
α	2
β	1
β	3

Set DIFFERENCE Operation

- Notation: $r - s$

- Defined as:

$$r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$$

- Set differences must be taken between *compatible* relations (*Union compatibility*).
  - r and s must have the *same number of attributes*.
  - Attribute domains of r and s must be compatible.

- It is a binary operation

- *No Commutative*

$$r - s \neq s - r$$

Set DIFFERENCE Operation - Example

- Relation *r, s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r - s$

A	B
α	1
β	1

Cartesian Product Operation

- Notation: $r \times s$

- Defined as:

$$r \times s = \{tq \mid t \in r \textbf{ and } q \in s\}$$

- Assume that attributes of r (R) and s (S) are disjoint; that is, $R \cap S = \varnothing$).

- If attributes of r (R) and s (S) are not disjoint, then renaming must be used.

- Do *not* have to be union compatible operation

- The result of $R(A_1, A_2, \ldots, A_n) \times S(B_1, B_2, \ldots, B_m)$ is a relation Q with $n + m$ attributes $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$ in that order. The resulting relation Q has one tuple for each combination of tuples—one from R and one from S. Hence, if R has n_R tuples and S has n_S tuples, then $R \times S$ will have $n_R \times n_S$ tuples.

- The sequences of this operation:
  - creating tuples with the combined attribute of two relations
  - SELECTing only related tuples from the two relations by specifying an appropriate selection conditions.

Cartesian Product Operation - Example

- Relations *r, s*:

A	B
α	1
β	2

r

C	D	E
α	10	+
β	10	+
β	20	–
γ	10	–

s

- $r \times s$

A	B	C	D	E
α	1	α	10	+
α	1	β	10	+
α	1	β	20	–
α	1	γ	10	–
β	2	α	10	+
β	2	β	10	+
β	2	β	20	–
β	2	γ	10	–

Composition of Operations

- Can build expressions using multiple operations
- Example: $\sigma_{A=C}(r \times s)$
  - $r \times s$

A	B	C	D	E
α	1	α	10	+
α	1	β	10	+
α	1	β	20	–
α	1	γ	10	–
β	2	α	10	+
β	2	β	10	+
β	2	β	20	–
β	2	γ	10	–

  - $\sigma_{A=C}(r \times s)$

A	B	C	D	E
α	1	α	10	+
β	2	β	10	+
β	2	β	20	–

RENAME Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.

  Example:

  $$\rho_{\alpha} (E)$$

  returns the expression E under the name α

  If a relational-algebra expression E has n attributes, then

  $$\rho_{\alpha(A_1, A_2, ..., A_n)} (E)$$

  returns the result of expression E under the name α, and with the attributes renamed to $A_1, A_2, ..., A_n$.

Banking Example

BRANCH (Branch-name, Branch-city, Assets)

CUSTOMER (Customer-name, Customer-street, Customer-city)

ACCOUNT (Branch-name, Account-number, Balance)

LOAN (Branch-name, Loan-number, Amount)

DEPOSITOR (Customer-name, Account-number)

BORROWER (Customer-name, Loan-number)

Banking Example Queries

- Find all loans of over $1200

- List the loan number for each loan of an amount greater than $1200

- Retrieve the names of all customers who have a loan, an account, or both, from the bank.

- Find the names of all customers who have a loan and an account at bank.

Banking Example Queries (Cont.)

- List the names of all customers who have a loan at the Tucson branch.

- Find the names of all customers who have a loan at the Tucson branch but do not have an account at any branch of the bank.

- Retrieve the names of all customers who have a loan at the Tucson branch.
  - Query 1

  - Query 2

Banking Example Queries (Cont.)

- Find the largest account balance
  - Rename *account* relation as *d*
  - The query is:

Formal Definition

- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:

  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p (E_1)$, P is a predicate on attributes in E_1
  - $\prod_s (E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_x (E_1)$, x is the new name for the result of E_1

Additional Operations

- We define additional operations that do not add any power to the relational algebra, but that simplify common queries.

  - Set INTERSECTION
  - NATURAL JOIN
  - DIVISION
  - ASSIGNMENT

Set INTERSECTION Operation

- Notation: $r \cap s$

- Defined as:

$$r \cap s = \{t \mid t \in r \textbf{ and } t \in s\}$$

- Assume (*Union compatibility*):
  - r, s have the same number of attributes
  - Attributes of r and s are compatible

- Note: $r \cap s = r - (r - s)$

- It is a binary operation.

- *Commutative*

$$r \cap s = s \cap r$$

- *Associative*

$$r \cap (s \cap t) = (r \cap s) \cap t$$

Set INTERSECTION Operation – Example

- Relations *r, s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

NATURAL JOIN Operation

- Notation: $r \bowtie s$

- Let r and s be relations on schemas R and S respectively. The result is a relation on schema $R \cup S$ which is obtained by considering each pair of tuples t_r from r and t_s from s.

- If t_r and t_s have the same value on each of the attributes in $R \cap S$, a tuple t is added to the result, where
  - t has the same value as t_r on r
  - t has the same value as t_s on s

- Example: $R = (A, B, C, D)$
  $S = (E, B, D)$
  - Result schema = (A, B, C, D, E)

- $r \bowtie s$ is defined as:

$$\Pi_{r.A,\, r.B,\, r.C,\, r.D,\, s.E} (\sigma_{r.B = s.B \wedge r.D = s.D} (r \times s))$$

NATURAL JOIN Operation (Cont.)

- The NATURAL JOIN requires that the two join attributes (or each pair of join attributes) have the same name. If this is not the case, a *renaming* operation is applied.

- In general, NATURAL JOIN is performed by equating *all* attribute pairs that have the same name in the two relations.

- It is a binary operation.

- In general, if r has t_r tuples and s has t_s tuples, the result of a JOIN operation $r \bowtie s$ will have between zero and $t_r \times t_s$ tuples.
  - If *no combination of tuples satisfies the join condition*, the result of a JOIN is an *empty relation with zero tuples*.
  - If *there is no common attribute to satisfy*, *all combinations* of tuples qualify and the JOIN becomes a Cartesian Product.

- There are three types of JOIN operations:
  - Theta Join — A join operation with a general join condition (i.e., containing any comparison operator).
  - Equijoin — A join operation involving join conditions with equality comparisons only.
  - Natural Join — Same as Equijoin, except that the join attributes of the second relation are not included in the resulting relations.

NATURAL JOIN Operation – Example

- Relation r, s:

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

s

- $r \bowtie s$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

DIVISION Operation

- Notation: $r \div s$

- Suited to queries that include the phrase "for all."

- Let r and s be relations on schemas R and S respectively, where

$$R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$$
$$S = (B_1, \ldots, B_n)$$

  The result of $r \div s$ is a relation on schema

$$R - S = (A_1, \ldots, A_m)$$

- It is a binary operation, but the relation on which it is applied do not have to be *union compatible*.

DIVISION Operation (Cont.)

- Property
  - Let $q = r \div s$
  - Then q is the largest relation satisfying: $q \times s \subseteq r$

- Definition in terms of the basic algebra operation
  Let r (R) and s (S) be relations, and let $S \subseteq R$

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}\left(\left(\Pi_{R-S}(r) \times s \right) - \Pi_{R-S,S}(r) \right)$$

  To see why:
  - $\Pi_{R-S,S}(r)$ simply reorders attributes of r
  - $\Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$ gives those tuples t in $\Pi_{R-S}(r)$ such that for some tuple $u \in s$, $tu \notin r$.

- $r \div s$ produces a relation that has the attributes of R that are *not* attributes of S *and* includes all tuples in R in combination with every tuple from S.

DIVISION Operation – Example 1

- Relation r, s:

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
δ	6
θ	1
θ	2

r

B
1
2

s

- $r \div s$

A
α
θ

DIVISION Operation – Example 2

- Relation r, s:

A	B	C	D	E
α	a	α	a	1
α	a	γ	a	1
α	a	γ	b	1
β	a	γ	a	1
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	a	β	b	1

r

D	E
a	1
b	1

s

- $r \div s$

A	B	C
α	a	γ
γ	a	γ

ASSIGNMENT Operation

- The assignment operation ($\leftarrow$) provides a convenient way to express complex queries; write query as sequential program consisting of a series of assignments followed by an expression whose value is displayed as the result of the query.

- Assignment must always be made to a temporary relation variable.

- Example: write $r \div s$ as

$$
\begin{array}{lll}
temp1 & \leftarrow & \Pi_{R-S}(r) \\
temp2 & \leftarrow & \Pi_{R-S}((temp1 \times s) - \Pi_{R-S,S}(r)) \\
result & = & temp1 - temp2
\end{array}
$$

  - The result to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$.

  - May use variable in subsequent expressions.

Banking Example Queries (Revisited)

- Find all customers who have an account from at least the "Downtown" and "Uptown" branches.

- Find all customers who have an account at all branches located in Brooklyn.

Extended Relational-Algebra-Operations

- OUTER JOIN
- Aggregate Functions

OUTER JOIN

- An extension of the join operation that avoids loss of information.

- Computes the join and then adds tuples from one relation that do not match tuples in the other relation to the result of the join.

- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist.
  - All comparisons involving *null* are **false** by definition

- LEFT OUTER JOIN ($⟕$) — keeps every tuple in the *first* or left relation *R* in $R ⟕ S$; if no matching tuple is found in *S*, then the attributes of *S* in the result are filled or "padded" with *null* values.

- RIGHT OUTER JOIN ($⟖$) — keeps every tuple in the *second* or right relation *S* in the result of $R ⟖ S$.

- FULL OUTER JOIN ($⟗$) — keeps all tuples in both the left and the right relations in the result of $R ⟗ S$ when no matching tuples are found, padding them with null values as needed.

OUTER JOIN – Example

- Relation LOAN

Branch-name	Loan-number	Amount
Downtown	L-170	3000
Redwood	L-230	4000
Tucson	L-260	1700

- Relation BORROWER

Customer-name	Loan-number
Jones	L-170
Smith	L-230
Lee	L-155

OUTER JOIN – Example (Cont.)

- LOAN $\bowtie$ BORROWER

Branch-name	Loan-number	Amount	Customer-name
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith

- LOAN ⟕ BORROWER

Branch-name	Loan-number	Amount	Customer-name	Loan-number
Downtown	L-170	3000	Jones	L-170
Redwood	L-230	4000	Smith	L-230
Tucson	L-260	1700	*null*	*null*

- LOAN ⟖ BORROWER

Branch-name	Loan-number	Amount	Customer-name	Loan-number
Downtown	L-170	3000	Jones	L-170
Redwood	L-230	4000	Smith	L-230
null	L-155	*null*	Lee	L-155

- LOAN ⟗ BORROWER

Branch-name	Loan-number	Amount	Customer-name
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith
Tucson	L-260	1700	*null*
null	L-155	*null*	Lee

Aggregate Functions

- Aggregation operator ϑ takes a collection of values and returns a single value as a result.

AVG:	average value
MIN:	minimum value
MAX:	maximum value
SUM:	sum of values
COUNT:	number of values

$$_{G_1, G_2, \ldots, G_n} \vartheta_{F_1\ A_1,\ F_2\ A_2, \ldots,\ F_m\ A_m} (E)$$

- E is any relational-algebra expression
- $G_1, G_2, \ldots, G_n$ is a list of attributes on which to group
- F_i is an aggregate function
- A_i is an attribute name

Aggregate Function – Example 1

- Relation *r*:

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

- $\mathbf{SUM}_c(r)$

sum-C
27

Aggregate Function – Example 2

- Relation ACCOUNT grouped by Branch-name:

Branch-name-	Account-number	Balance
Tucson	A-102	400
Tucson	A-201	900
Pittsburgh	A-217	750
Pittsburgh	A-215	900
Redwood	A-222	700

- $_{\text{Branch-name}}\,\vartheta\,_{\textbf{SUM}\ \text{Balance}}$(ACCOUNT)

Branch-name-	Balance
Tucson	1300
Pittsburgh	1650
Redwood	700

Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating

- All these operations are expressed using the ASSIGNMENT operator.

Deletion

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.

- Can delete only whole tuples: cannot delete values on only particular attributes

- A deletion is expressed in relational algebra by:

$$r \leftarrow r - E$$

  where r is a relation and E is a relational algebra query.

- Examples:
  - Delete all account records in the Tucson branch.
  - Delete all loan records with amount in the range 0 to 50.
  - Delete all accounts at branches located in Pittsburgh.

Insertion

- To insert data into a relation, we either:
  - specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted

- In relational algebra, an insertion is expressed by:

$$r \leftarrow r \cup E$$

  where r is a relation and E is a relational algebra expression.

- The insertion of a single tuple is expressed by letting E be a constant relation containing one tuple

- Examples:
  - Insert information in the database specifying that John has \$1200 in account A-973 at the Tucson branch.

  - Provide as a gift for all loan customers in the Tucson branch, a \$100 savings account. Let the loan number serve as the account number for the new savings account.

Updating

- A mechanism to change a value in tuple without changing *all* values in the tuple

- Use the generalized projection operator to do this task

$$r \leftarrow \Pi_{F_1, F_2, \ldots, F_n}(r)$$

  - Each F_i is either ith attribute of r, if the ith attribute is not updated, or,
  - F_i is an expression, if the attribute is to be updated, involving only constants and the attributes of r, which gives the new value for the attribute

- Examples:
  - Make interest payments by increasing all balances by 5 percent.
  - Pay all accounts with balance over $10,000 6 percent interest and pay all others 5 percent.

Views

- In some cases, it is not desirable for all users to see the entire logical model (i.e., all the actual stored in the database) — *security* consideration.

- Consider a person who needs to know a customer's loan number but has no need to see the loan amount. This person should see a relation described, in the relational algebra, by

$$\Pi_{\text{Customer-name, Loan-number}} (\text{BORROWER} \bowtie \text{LOAN})$$

- Any relation that is not part of the conceptual model but is made visible to a user as a "virtual relation" is called a *view*.

View Definition

- A view is defined using the **CREATE VIEW** statement which has the form

  CREATE VIEW *v* **AS** <query expression>

  where <query expression> is any legal relational algebra query expression. The view is represented by *v.*

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.

- View definition is not the *same* as creating a new relation by evaluating the query expression. Rather, a view definition causes the saving of an expression to be substituted into queries using the view — The data will be *materialized* at the time the view (query definitions) is executed. Thus, the view is kept up to date.

- Examples:
  - Consider the view (named ALL-CUSTOMER) consisting of branches and their customers.

  - We can find all customers of the Tucson branch by writing:

Update Through Views

- Database modifications expressed as views must be translated to modifications of the relations in the database.

- *Single relation* — Consider the person who needs to see all loan in the LOAN relation except Loan-amount. The view given to the person, BRANCH-LOAN, is defined as:

  CREATE VIEW BRANCH-LOAN **AS**
  $\Pi_{\text{Branch-name, Loan-number}}$ (LOAN)

  Since we allow a view name to appear wherever a relation name is allowed, the person may write:

  BRANCH-LOAN $\leftarrow$ BRANCH-LOAN $\cup$ {("Tucson", L-37)}

  - The previous insertion must be represented by an insertion into the actual relation LOAN from which the view BRANCH-LOAN is constructed.

  - An insertion into LOAN requires a value for Amount. The insertion can dealt with by either
    - rejecting the insertion
    - inserting a tuple ("Tucson", L-37, *null*) into the LOAN relation

Update Through Views (Cont.)

- *Multiple relation* — Consider the person who needs to see all the customers and their amount. In this case, we need two relations to obtain the proper information. The view, LOAN-INFO, is defined as:

  CREATE VIEW LOAN-INFO **AS**
  $\Pi_{\text{Customer-name, Amount}}$ (BORROWER $\bowtie$ LOAN)

  If a person insert the following information through the view:

  LOAN-INFO $\leftarrow$ LOAN-INFO $\cup$ {("Johnson", 1900)}

  - The only possible method of inserting this tuple into the relations is to insert ("Johnson", *null*) into BORROWER and (*null*, *null*, 1900) into LOAN.

  - This operation does not have the desired effect, because LOAN-INFO couldn't include the tuple ("Johnson", 1900). The required attributes (Loan-number from both relations) for the NATURAL JOIN contain *null* values. Thus, the LOAN-INFO relation cannot materialize this data.

- In general, most database does not support view updates for views that are defined from multiple relations; some databases support view modification for views that are defined from a single relation.

Views Defined Using Other Views

- One view may be used in the expression defining another view.

- A view relation v_1 is said to *depend directly on* view relation v_2, if v_2 is used in the expression defining v_1.

- A view relation v_1 is said to *depend on* view relation v_2, if and only if there is a path in the dependency graph from v_2 to v_1.

- A view relation v is said to *recursive* if it depends on itself.

- View Expansion:
  - A way to define the meaning of views defined in terms of other views.
  - Let view v_1 be defined by an expression e_1 that may itself contain uses of view relations.
  - View expansion of an expression repeats the following replacement step:

    repeat
        Find any view relation v_i in e_1
        Replace the view relation v_i by
            the expression defining v_i
    until no more view relations are present in e_1

  - As long as the view definitions are not recursive, this loop will terminate.

Company Example (Additional Exercise)

EMPLOYEE (SSN, Name, Salary, Supervisor-SSN, Dnumber)

DEPARTMENT (Dnumber, Dname, Mgr-SSN, Mgr-start-date)

PROJECT (Pnumber, Pname, Dnumber)

WORKS-ON (SSN, Pnumber, Hours)

Company Example Queries

SELECT Operation

- Retrieve all employees who work in department 4 or whose salary is greater than $30,000.

- Retrieve all departments whose managers' starting date is between 01-Jan-85 and 31-Dec-85.

- Retrieve all projects managed by department 5 or 6.

- Retrieve all employees who spend more than 10 hours in a project.

PROJECT Operation

- List each employee's name and salary.

- List each department's name and the SSN of the manager.

- Retrieve the project's name and number for each project managed by department 5 or 6.

- Retrieve the SSN of each employee who spends more than 10 hours in a project.

UNION Operation

- List the SSN of an employee whose salary is either greater than \$30,000 or who are working on project 6.

- List the SSNs of the department 3's employees and the manager of the department 3.

DIFFERENCE Operation

- List the SSN of an employee whose salary is either greater than \$30,000 and who are not working on project 6.

- List the SSNs of the department 3's employees who salary is lower than \$30,000.

INTERSECTION Operation

- List the SSN of an employee whose salary is greater than $30,000 and who are working on project 6.

- List the SSNs of the managers of the departments who are working on project 8.

Cartesian Product Operation

- List all combination of employees and departments.

- List each employee's name and his/her department name.

- List each employee's name and salary who is working on project 3.

JOIN Operation

- List each employee’s name and his/her department name.

- List each employee’s name and salary who is working on project 3.

DIVISION Operation

- List the SSN of each employee who is working on all projects.

RENAME Operation

- List the names of all employees who are in the same department as Smith.

Database Modification

- Delete the employee whose SSN = 123456789.

- Insert a project whose name is Medical–DB, whose number is 100, and which is managed by MIS department (department number is 15).

- Update the salary of all employees by increasing 5%.

- Update the salary of the employees who are working for department 15 by increasing 5%.