Chapter 3: Mapping Conceptual Design into a Logical Design

3.1 Relational Database Design using ER-to-Relational mapping

Procedure to create a relational schema from an Entity-Relationship (ER)

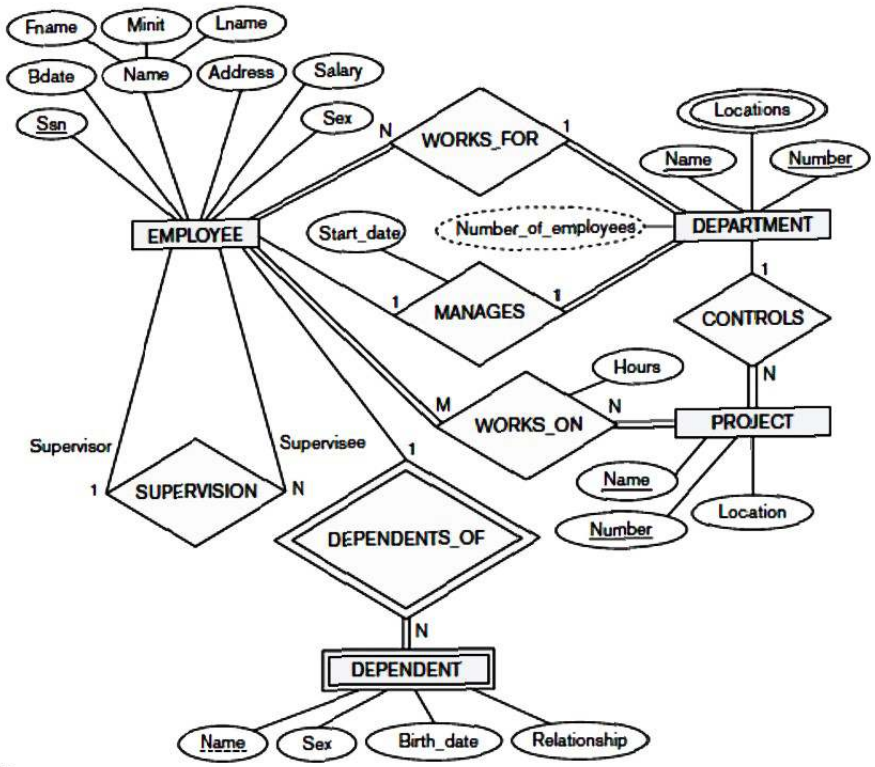


Fig 3.1: ER diagram of company database

Step 1: Mapping of Regular Entity Types

- For each regular entity type, create a relation R that includes all the simple attributes of E
- Include only the simple component attributes of a composite attribute
- Choose one of the key attributes of E as the primary key for R
- If the chosen key of E is a composite, then the set of simple attributes that form it will together form the primary key of R.

- If multiple keys were identified for E during the conceptual design, the information describing the attributes that form each additional key is kept in order to specify secondary (unique) keys of relation R
- In our example-COMPANY database, we create the relations EMPLOYEE, DEPARTMENT, and PROJECT
- we choose Ssn, Dnumber, and Pnumber as primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT, respectively
- The relations that are created from the mapping of entity types are called **entity relations** because each tuple represents an entity instance.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary

DEPARTMENT

Dname	Dnumber

- If multiple keys were identified for E during the conceptual design, the information describing the attributes that form each additional key is kept in order to specify secondary (unique) keys of relation R
- In our example-COMPANY database, we create the relations EMPLOYEE, DEPARTMENT, and PROJECT
- we choose Ssn, Dnumber, and Pnumber as primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT, respectively
- The relations that are created from the mapping of entity types are called **entity relations** because each tuple represents an entity instance.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary

DEPARTMENT

Dname	Dnumber

PROJECT

Pname	Pnumber	Plocation

Step 2: Mapping of Weak Entity Types

- For each weak entity type, create a relation R and include all simple attributes of the entity type as attributes of R
- Include primary key attribute of owner as foreign key attributes of R
- In our example, we create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT
- We include the primary key Ssn of the EMPLOYEE relation—which corresponds to the owner entity type—as a foreign key attribute of DEPENDENT; we rename it as Essn
- The primary key of the DEPENDENT relation is the combination {Essn,Dependent_name}, because Dependent_name is the partial key of DEPENDENT
- It is common to choose the propagate (CASCADE) option for the referential triggered action on the foreign key in the relation corresponding to the weak entity type, since a weak entity has an existence dependency on its owner entity.
- This can be used for both ON UPDATE and ON DELETE.

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Step 3: Mapping of Binary 1:1 Relationship Types

- For each binary 1:1 relationship type *R* in the ER schema, identify the relations *S* and *T* that correspond to the entity types participating in *R*
- There are three possible approaches:
  - foreign key approach
  - merged relationship approach
  - crossreference or relationship relation approach

1. The foreign key approach

- Choose one of the relations—*S*, say—and include as a foreign key in *S* the primary key of *T*.
- It is better to choose an entity type with *total participation* in *R* in the role of *S*
- Include all the simple attributes (or simple components of composite attributes) of the 1:1 relationship type *R* as attributes of *S*.
- In our example, we map the 1:1 relationship type by choosing the participating entity type DEPARTMENT to serve in the role of *S* because its participation in the MANAGES relationship type is total
- We include the primary key of the EMPLOYEE relation as foreign key in the DEPARTMENT relation and rename it Mgr_ssn.
- We also include the simple attribute Start_date of the MANAGES relationship type in the DEPARTMENT relation and rename it Mgr_start_date

2. Merged relation approach:

- merge the two entity types and the relationship into a single relation
- This is possible when *both participations are total,* as this would indicate that the two

2. **Merged relation approach:**
   - merge the two entity types and the relationship into a single relation
   - This is possible when *both participations are total,* as this would indicate that the two tables will have the exact same number of tuples at all times.
3. **Cross-reference or relationship relation approach:**
   - set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types.
   - required for binary M:N relationships
   - The relation R is called a relationship relation (or sometimes a lookup table), because each tuple in R represents a relationship instance that relates one tuple from S with one tuple from T
   - The relation R will include the primary key attributes of S and T as foreign keys to S and T.
   - The primary key of R will be one of the two foreign keys, and the other foreign key will be a unique key of R.

- The drawback is having an extra relation, and requiring an extra join operation when combining related tuples from the tables.

Step 4: Mapping of Binary 1:N Relationship Types

- For each regular binary 1:N relationship type *R*, identify the relation *S* that represents the participating entity type at the *N-side* of the relationship type.
- Include as foreign key in *S* the primary key of the relation *T* that represents the other entity type participating in *R*
- Include any simple attributes (or simple components of composite attributes) of the 1:N relationship type as attributes of *S*
- In our example, we now map the 1:N relationship types WORKS_FOR, CONTROLS, and SUPERVISION
- For WORKS_FOR we include the primary key Dnumber of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it Dno.
- For SUPERVISION we include the primary key of the EMPLOYEE relation as foreign key in the EMPLOYEE relation itself—because the relationship is recursive—and call it Super_ssn.
- The CONTROLS relationship is mapped to the foreign key attribute Dnum of PROJECT, which references the primary key Dnumber of the DEPARTMENT relation.

Step 5: Mapping of Binary M:N Relationship Types

- For each binary M:N relationship type
  - Create a new relation S
  - Include primary key of participating entity types as foreign key attributes in S
  - Include any simple attributes of M:N relationship type
- In our example, we map the M:N relationship type WORKS_ON by creating the relation WORKS_ON.We include the primary keys of the PROJECT and EMPLOYEE relations as foreign keys in WORKS_ON and rename them Pno and Essn, respectively.
- We also include an attribute Hours in WORKS_ON to represent the Hours attribute of the relationship type.
- The primary key of the WORKS_ON relation is the combination of the foreign key attributes {Essn, Pno}.

WORKS_ON

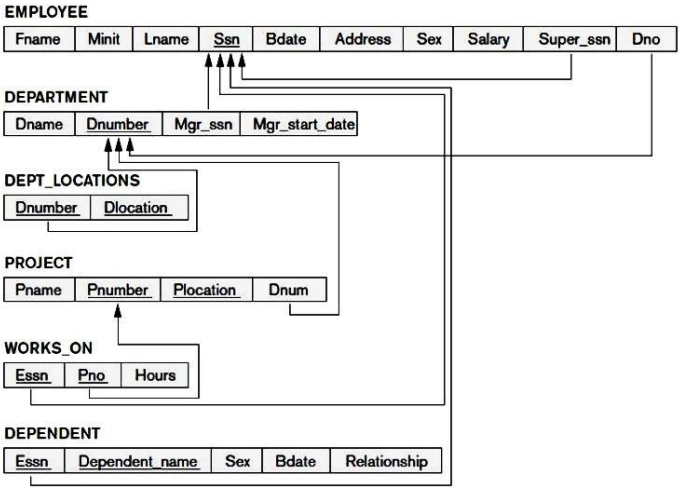
Essn	Pno	Hours

- The propagate (CASCADE) option for the referential triggered action should be specified

- The propagate (CASCADE) option for the referential triggered action should be specified on the foreign keys in the relation corresponding to the relationship R, since each relationship instance has an existence dependency on each of the entities it relates. This can be used for both ON UPDATE and ON DELETE.

Step 6: Mapping of Multivalued Attributes

- For each multivalued attribute
  - Create a new relation
  - Primary key of *R* is the combination of *A* and *K*
  - If the multivalued attribute is composite, include its simple components
- In our example, we create a relation DEPT_LOCATIONS
- The attribute Dlocation represents the multivalued attribute LOCATIONS of DEPARTMENT, while Dnumber—as foreign key—represents the primary key of the DEPARTMENT relation.
- The primary key of DEPT_LOCATIONS is the combination of {Dnumber, Dlocation}
- A separate tuple will exist in DEPT_LOCATIONS for each location that a department has
- The propagate (CASCADE) option for the referential triggered action should be specified on the foreign key in the relation *R* corresponding to the multivalued attribute for both ON UPDATE and ON DELETE.



Step 7: Mapping of *N*-ary Relationship Types

- For each *n*-ary relationship type *R*
  - Create a new relation *S* to represent *R*
  - Include primary keys of participating entity types as foreign keys
  - Include any simple attributes as attributes
- The primary key of *S* is usually a combination of all the foreign keys that reference the relations representing the participating entity types.
- For example, consider the relationship type SUPPLY.This can be mapped to the relation SUPPLY whose primary key is the combination of the three foreign keys {Sname, Part_no, Proj_name}.

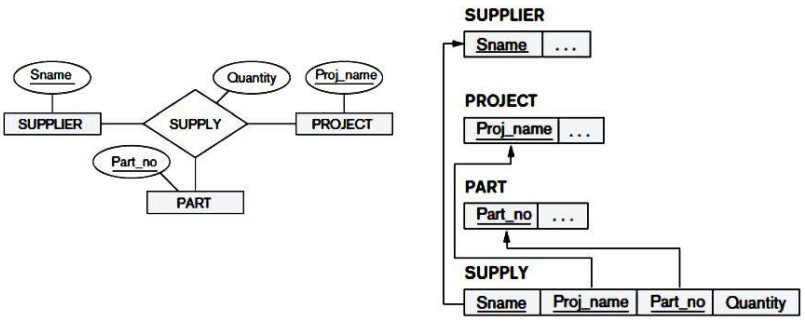


Figure 3.2: Mapping the *n*-ary relationship type SUPPLY

Delete the EMPLOYEE tuple with Ssn = '333445555'

Result: This deletion will result in even worse referential integrity violations, because the tuple involved is referenced by tuples from the EMPLOYEE, DEPARTMENT, WORKS_ON, and DEPENDENT relations.

Several options are available if a deletion operation causes a violation

1. restrict - is to reject the deletion
2. cascade, is to attempt to cascade (or propagate) the deletion by deleting tuples that reference the tuple that is being deleted

3. Set null or set default - is to modify the referencing attribute values that cause the violation; each such value is either set to NULL or changed to reference another default valid tuple.

1.3.3 The Update Operation

The Update (or Modify) operation is used to change the values of one or more attributes in a tuple (or tuples) of some relation *R*. It is necessary to specify a condition on the attributes of the relation to select the tuple (or tuples) to be modified.

Examples:

1. Operation:
   Update the salary of the EMPLOYEE tuple with Ssn = '999887777' to 28000.
   Result: Acceptable.
2. Operation:
   Update the Dno of the EMPLOYEE tuple with Ssn = '999887777' to 7.
   Result: Unacceptable, because it violates referential integrity.
3. Operation:
   Update the Ssn of the EMPLOYEE tuple with Ssn = '999887777' to '987654321'.
   Result: Unacceptable, because it violates primary key constraint by repeating a value that already exists as a primary key in another tuple; it violates referential integrity constraints because there are other relations that refer to the existing value of Ssn

Updating an attribute that is neither part of a primary key nor of a foreign key usually causes no problems; the DBMS need only check to confirm that the new value is of the correct data type and domain.

1.3.4 The Transaction Concept

A **transaction** is an executing program that includes some database operations, such as reading from the database, or applying insertions, deletions, or updates to the database. At the end of the transaction, it must leave the database in a valid or consistent state that satisfies all the constraints specified on the database schema A single transaction may involve any number of retrieval operations and any number of update operations. These retrievals and updates will together form an atomic unit of work against the database.For example, a transaction to apply a bank withdrawal will typically read the user account record, check if there is a sufficient balance, and then update the record by the withdrawal amount.

Chapter 2: Relational Algebra

2.1 Introduction

Relational algebra is the basic set of operations for the relational model. These operations enable a user to specify basic retrieval requests as relational algebra expressions. The result of an operation is a new relation, which may have been formed from one or more input relations.

The relational algebra is very important for several reasons

- First, it provides a formal foundation for relational model operations.
- Second, and perhaps more important, it is used as a basis for implementing and optimizing

A **transaction** is an executing program that includes some database operations, such as reading from the database, or applying insertions, deletions, or updates to the database. At the end of the transaction, it must leave the database in a valid or consistent state that satisfies all the constraints specified on the database schema A single transaction may involve any number of retrieval operations and any number of update operations. These retrievals and updates will together form an atomic unit of work against the database.For example, a transaction to apply a bank withdrawal will typically read the user account record, check if there is a sufficient balance, and then update the record by the withdrawal amount.

Chapter 2: Relational Algebra

2.1 Introduction

Relational algebra is the basic set of operations for the relational model. These operations enable a user to specify basic retrieval requests as relational algebra expressions. The result of an operation is a new relation, which may have been formed from one or more input relations.

The relational algebra is very important for several reasons

- First, it provides a formal foundation for relational model operations.
- Second, and perhaps more important, it is used as a basis for implementing and optimizing queries in the query processing and optimization modules that are integral parts of relational database management systems (RDBMSs)
- Third, some of its concepts are incorporated into the SQL standard query language for RDBMSs

2.2 Unary Relational Operations: SELECT and PROJECT

2.2.1 The SELECT Operation

The SELECT operation denoted by σ (sigma) is used to select a subset of the tuples from a relation based on a selection condition. The selection condition acts as a filter that keeps only those tuples that satisfy a qualifying condition. Alternatively, we can consider the SELECT operation to *restrict* the tuples in a relation to only those tuples that satisfy the condition.

The SELECT operation can also be visualized as a *horizontal partition* of the relation into two sets of tuples—those tuples that satisfy the condition and are selected, and those tuples that do not satisfy the condition and are discarded.

In general, the select operation is denoted by

$$\sigma_{\text{<selection condition>}}(R)$$

where,

- the symbol σ is used to denote the select operator
- the selection condition is a Boolean (conditional) expression specified on the attributes of relation R
- tuples that make the condition true are selected
  - appear in the result of the operation
- tuples that make the condition false are filtered out
  - discarded from the result of the operation

The Boolean expression specified in <selection condition> is made up of a number of clauses of the form:

<attribute name> <comparison op> <constant value>

or

<attribute name> <comparison op> <attribute name>

where

<attribute name> is the name of an attribute of *R*.

The general RENAME operation when applied to a relation R of degree n is denoted by any of the following three forms:

1. $\rho_{S(B1, B2, ..., Bn)}(R)$ — ρ (rho) – RENAME operator
2. $\rho S(R)$ — S – new relation name
3. $\rho_{(B1, B2, ...Bn)}(R)$ — $B_1, B_2, ... B_n$- new attribute names

The first expression renames both the relation and its attributes. Second renames the relation only and the third renames the attributes only.If the attributes of R are (A_1, A_2, ..., A_n) in that order, then each A_i is renamed as B_i.

Renaming in SQL is accomplished by aliasing using AS, as in the following example:

```
SELECT E.Fname AS First_name,
       E.Lname AS Last_name,
       E.Salary AS Salary
       FROM EMPLOYEE AS E
       WHERE E.Dno=5,
```

2.3 Relational Algebra Operations from Set Theory

2.3.1 The UNION, INTERSECTION, and MINUS Operations

- **UNION:** The result of this operation, denoted by $R \cup S$, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.
- **INTERSECTION:** The result of this operation, denoted by $R \cap S$, is a relation that includes all tuples that are in both R and S.
- **SET DIFFERENCE (or MINUS):** The result of this operation, denoted by $R - S$, is a relation that includes all tuples that are in R but not in S.

Example: Consider the the following two relations: STUDENT & INSTRUCTOR

STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

STUDENT ∪ INSTRUCTOR

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

STUDENT ∩ INSTRUCTOR

Fn	Ln
Susan	Yao
Ramesh	Shah

STUDENT – INSTRUCTOR

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR – STUDENT

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

STUDENT – INSTRUCTOR

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR – STUDENT

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Example: To retrieve the Social Security numbers of all employees who either work in department 5 or directly supervise an employee who works in department 5

$$DEP5\_EMPS \leftarrow \sigma_{Dno=5}(EMPLOYEE)$$

$$RESULT1 \leftarrow \pi_{Ssn}(DEP5\_EMPS)$$

$$RESULT2(Ssn) \leftarrow \pi_{Super\_ssn}(DEP5\_EMPS)$$

$$RESULT \leftarrow RESULT1 \cup RESULT2$$

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	gender	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

Single relational algebra expression:

$$\textbf{Result} \leftarrow \pi_{Ssn}(\sigma_{Dno=5}(\textbf{EMPLOYEE})) \cup \pi_{Super\_ssn}(\sigma_{Dno=5}(\textbf{EMPLOYEE}))$$

UNION, INTERSECTION and SET DIFFERENCE are binary operations; that is, each is applied to two sets (of tuples). When these operations are adapted to relational databases, the two relations on which any of these three operations are applied must have the same type of tuples; this condition has been called **union compatibility or type compatibility.**

Two relations $R(A_1, A_2, ..., A_n)$ and $S(B_1, B_2, ..., B_n)$ are said to be union compatible (or type compatible) if they have the same degree n and if $dom(A_i) = dom(B_i)$ for $1 \leq i \leq n$. This means that the two relations have the same number of attributes and each corresponding pair of attributes has the same domain.

Both UNION and INTERSECTION are *commutative operations*; that is,

$$R \cup S = S \cup R \text{ and } R \cap S = S \cap R$$

Both UNION and INTERSECTION can be treated as *n*-ary operations applicable to any number of relations because both are also *associative operations;* that is,

$$R \cup (S \cup T) = (R \cup S) \cup T \text{ and } (R \cap S) \cap T = R \cap (S \cap T)$$

The MINUS operation is *not commutative;* that is, in general,

$$R - S \neq S - R$$

INTERSECTION can be expressed in terms of union and set difference as follows:

$$R \cap S = ((R \cup S) - (R - S)) - (S - R)$$

In SQL, there are three operations—UNION, INTERSECT, and EXCEPT—that correspond to the set operations

Chapter 2: Relational Algebra

2.1 Introduction

Relational algebra is the basic set of operations for the relational model. These operations enable a user to specify basic retrieval requests as relational algebra expressions. The result of an operation is a new relation, which may have been formed from one or more input relations.

The relational algebra is very important for several reasons

- First, it provides a formal foundation for relational model operations.
- Second, and perhaps more important, it is used as a basis for implementing and optimizing queries in the query processing and optimization modules that are integral parts of relational database management systems (RDBMSs)
- Third, some of its concepts are incorporated into the SQL standard query language for RDBMSs

2.2 Unary Relational Operations: SELECT and PROJECT

2.2.1 The SELECT Operation

The SELECT operation denoted by σ (sigma) is used to select a subset of the tuples from a relation based on a selection condition. The selection condition acts as a filter that keeps only those tuples that satisfy a qualifying condition. Alternatively, we can consider the SELECT operation to *restrict* the tuples in a relation to only those tuples that satisfy the condition.

The SELECT operation can also be visualized as a *horizontal partition* of the relation into two sets of tuples—those tuples that satisfy the condition and are selected, and those tuples that do not satisfy the condition and are discarded.

In general, the select operation is denoted by

$$\sigma_{\text{<selection condition>}}(R)$$

where,

- the symbol σ is used to denote the select operator
- the selection condition is a Boolean (conditional) expression specified on the attributes of relation R
- tuples that make the condition true are selected
  - appear in the result of the operation
- tuples that make the condition false are filtered out
  - discarded from the result of the operation

The Boolean expression specified in <selection condition> is made up of a number of clauses of the form:

<attribute name> <comparison op> <constant value>

or

<attribute name> <comparison op> <attribute name>

where

<attribute name> is the name of an attribute of *R*,

<comparison op> is one of the operators {=, <, ≤, >, ≥, ≠}, and

<constant value> is a constant value from the attribute domain

Clauses can be connected by the standard Boolean operators *and*, *or*, and *not* to form a general selection condition

Examples:

1. Select the EMPLOYEE tuples whose department number is 4.

$$\sigma_{DNO = 4}(EMPLOYEE)$$

2. Select the employee tuples whose salary is greater than $30,000.

$$\sigma_{SALARY > 30,000}(EMPLOYEE)$$

3. Select the tuples for all employees who either work in department 4 and make over $25,000 per year, or work in department 5 and make over $30,000

$$\sigma_{(Dno=4 \text{ AND } Salary>25000)} \text{ OR } _{(Dno=5 \text{ AND } Salary>30000)}(EMPLOYEE)$$

The result of a SELECT operation can be determined as follows:

- The <selection condition> is applied independently to each individual tuple *t* in R

Examples:

1. Select the EMPLOYEE tuples whose department number is 4.

$$\sigma_{DNO = 4}(EMPLOYEE)$$

2. Select the employee tuples whose salary is greater than $30,000.

$$\sigma_{SALARY > 30,000}(EMPLOYEE)$$

3. Select the tuples for all employees who either work in department 4 and make over $25,000 per year, or work in department 5 and make over $30,000

$$\sigma_{(Dno=4 \text{ AND } Salary>25000)} \text{ OR } _{(Dno=5 \text{ AND } Salary>30000)}(EMPLOYEE)$$

The result of a SELECT operation can be determined as follows:

- The <selection condition> is applied independently to each individual tuple t in R
- If the condition evaluates to TRUE, then tuple *t* is selected.All the selected tuples appear in the result of the SELECT operation
- The Boolean conditions AND, OR, and NOT have their normal interpretation, as follows:

  - (cond1 AND cond2) is TRUE if both (cond1) and (cond2) are TRUE; otherwise,it is FALSE.

  - (cond1 OR cond2) is TRUE if either (cond1) or (cond2) or both are TRUE; otherwise, it is FALSE.

  - (NOT cond) is TRUE if cond is FALSE; otherwise, it is FALSE.

The SELECT operator is unary; that is, it is applied to a single relation. The degree of the relation resulting from a SELECT operation is the same as the degree of R.The number of tuples in the resulting relation is always less than or equal to the number of tuples in R. That is,

$|\sigma_c (R)| \leq |R|$ for any condition C

The fraction of tuples selected by a selection condition is referred to as the selectivity of the condition.

The SELECT operation is commutative; that is,

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

Hence, a sequence of SELECTs can be applied in any order.we can always combine a cascade (or sequence) of SELECT operations into a single SELECT operation with a conjunctive (AND) condition; that is,

$$\sigma_{<cond1>}(\sigma<cond2>(...(\sigma_{<condn>}(R)) ...)) = \sigma_{<cond1>} \text{ AND}_{<cond2>} \text{ AND } ... \text{ AND } _{<condn>}(R)$$

In SQL, the SELECT condition is specified in the WHERE clause of a query.For example, the following operation:

$$\sigma_{Dno=4} \text{ AND } _{Salary>25000}(EMPLOYEE)$$

would to the following SQL query:

SELECT * FROM EMPLOYEE WHERE Dno=4 AND Salary>25000;

2.2.2 The PROJECT Operation

The PROJECT operation denoted by π (pi) selects certain columns from the table and discards the other columns. Used when we are interested in only certain attributes of a relation. The result of the PROJECT operation can be visualized as a vertical partition of the relation into two relations:

- one has the needed columns (attributes) and contains the result of the operation
- the other contains the discarded columns

The general form of the PROJECT operation is

$$\pi_{<attribute\ list>}(R)$$

where

π (pi) - symbol used to represent the PROJECT operation,

<attributelist> - desired sublist of attributes from the attributes of relation R.

The result of the PROJECT operation has only the attributes specified in <attribute list> in the same order as they appear in the list. Hence, its degree is equal to the number of attributes in <attribute list>

<attributelist> - desired sublist of attributes from the attributes of relation R.

The result of the PROJECT operation has only the attributes specified in <attribute list> in the same order as they appear in the list. Hence, its degree is equal to the number of attributes in <attribute list>

Example :

1. To list each employee's first and last name and salary we can use the PROJECT operation as follows:

$$\pi_{\text{Lname, Fname, Salary}}(\text{EMPLOYEE})$$

If the attribute list includes only nonkey attributes of R, duplicate tuples are likely to occur. The result of the PROJECT operation is a set of distinct tuples, and hence a valid relation. This is known as **duplicate elimination**.For example, consider the following PROJECT operation:

$$\pi_{\text{gender, Salary}}(\text{EMPLOYEE})$$

Lname	Fname	Salary
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

The tuple <'F', 25000> appears only once in resulting relation even though this combination of values appears twice in the EMPLOYEE relation.

The number of tuples in a relation resulting from a PROJECT operation is always less than or equal to the number of tuples in *R*. Commutativity does not hold on PROJECT

$$\pi_{\text{<list1>}}(\pi_{\text{<list2>}}(R)) = \pi_{\text{<list1>}}(R)$$

as long as <list2> contains the attributes in <list1>; otherwise, the left-hand side is an incorrect expression.

In SQL, the PROJECT attribute list is specified in the SELECT clause of a query. For example, the following operation:

$$\pi_{\text{gender, Salary}}(\text{EMPLOYEE})$$

would correspond to the following SQL query:

SELECT DISTINCT gender, Salary **FROM** EMPLOYEE

2.2.3 Sequences of Operations and the RENAME Operation

For most queries, we need to apply several relational algebra operations one after the other. Either we can write the operations as a single relational algebra expression by nesting the operations, or we can apply one operation at a time and create intermediate result relations. In the latter case, we must give names to the relations that hold the intermediate results.

For example, to retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a SELECT and a PROJECT operation. We can write a single relational algebra expression, also known as an **in-line expression**, as follows:

$$\pi_{\text{Fname, Lname, Salary}}(\sigma_{\text{Dno=5}}(\text{EMPLOYEE}))$$

Alternatively, we can explicitly show the sequence of operations, giving a name to each intermediate relation, as follows:

2.5 Additional Relational Operations

2.5.1 Generalized Projection

The generalized projection operation extends the projection operation by allowing functions of attributes to be included in the projection list. The generalized form can be expressed as:

$$\pi_{F1, F2, ..., Fn}(R)$$

where F_1, F_2, ..., F_n are functions over the attributes in relation R and may involve arithmetic operations and constant values.

The generalized projection helpful when developing reports where computed values have to be produced in the columns of a query result. For example,consider the relation EMPLOYEE (Ssn, Salary,Deduction, Years_service). A report may be required to show

Net Salary = Salary – Deduction,

Bonus = 2000 * Years_service, and

Tax = 0.25 * Salary.

generalized projection combined with renaming :

$$\text{REPORT} \leftarrow \rho_{(Ssn, Net\_salary, Bonus, Tax)}(\pi_{Ssn, Salary - Deduction, 2000 * Years\_service, 0.25 * Salary}(\text{EMPLOYEE})).$$

2.5.2 Aggregate Functions and Grouping

Aggregate functions are used in simple statistical queries that summarize information from the database tuples.Common functions applied to collections of numeric values include SUM,AVERAGE, MAXIMUM, and MINIMUM.The COUNT function is used for counting tuples or values. For example, retrieving the average or total salary of all employees or the total number of employee tuples.

Grouping the tuples in a relation by the value of some of their attributes and then applying an aggregate function independently to each group. For example , group EMPLOYEE tuples by Dno, so that each group includes the tuples for employees working in the same department. We can then list each Dno value along with, say, the average salary of employees within the department, or the number of employees who work in the department.

Aggregate function operation can be defined by using the symbol ℑ (script F) :

$$\text{<grouping attributes>}\ \Im\ \text{<function list>}\ (R)$$

Where ,

<grouping attributes> : list of attributes of the relation specified in *R*

<function list> : list of (<function> <attribute>) pairs.

<function> - such as SUM, AVERAGE, MAXIMUM, MINIMUM,COUNT

<attribute> is an attribute of the relation specified by R

The resulting relation has the grouping attributes plus one attribute for each element in the function list.

Example: To retrieve each department number, the number of employees in the department, and their average salary, while renaming the resulting attributes

$$\rho_{R(Dno, No\_of\_employees, Average\_sal)}(Dno\ \Im\ \text{COUNT}_{Ssn, \text{AVERAGE}\ Salary}\ (\text{EMPLOYEE}))$$

The aggregate function operation.

a. $\rho_{R(Dno, No\_of\_employees, Average\_sal)}(Dno\ \Im\ \text{COUNT}_{Ssn, \text{AVERAGE}\ Salary}(\text{EMPLOYEE}))$.

b. $Dno\ \Im\ \text{COUNT}_{Ssn, \text{AVERAGE}\ Salary}(\text{EMPLOYEE})$.

c. $\Im\ \text{COUNT}_{Ssn, \text{AVERAGE}\ Salary}(\text{EMPLOYEE})$.

R

(a)

Dno	No_of_employees	Average_sal
5	4	33250
4	3	31000
1	1	55000

(b)

Dno	Count_ssn	Average_salary
5	4	33250
4	3	31000
1	1	55000

Grouping the tuples in a relation by the value of some of their attributes and then applying an aggregate function independently to each group. For example , group EMPLOYEE tuples by Dno, so that each group includes the tuples for employees working in the same department. We can then list each Dno value along with, say, the average salary of employees within the department, or the number of employees who work in the department.

Aggregate function operation can be defined by using the symbol ℑ (script F) :

$$\langle \text{grouping attributes} \rangle \, \Im \, \langle \text{function list} \rangle \, (R)$$

Where ,

<grouping attributes> : list of attributes of the relation specified in *R*

<function list> : list of (<function> <attribute>) pairs.

<function> - such as SUM, AVERAGE, MAXIMUM, MINIMUM,COUNT

<attribute> is an attribute of the relation specified by R

The resulting relation has the grouping attributes plus one attribute for each element in the function list.

Example: To retrieve each department number, the number of employees in the department, and their average salary, while renaming the resulting attributes

$$\rho_{R(\text{Dno, No\_of\_employees, Average\_sal})}(\text{Dno } \Im \text{ COUNT Ssn, AVERAGE Salary } (\text{EMPLOYEE}))$$

The aggregate function operation.

a. $\rho_{R(\text{Dno, No\_of\_employees, Average\_sal})}(\text{Dno } \Im \text{ COUNT Ssn, AVERAGE Salary}(\text{EMPLOYEE}))$.

b. $\text{Dno } \Im \text{ COUNT Ssn, AVERAGE Salary}(\text{EMPLOYEE})$.

c. $\Im \text{ COUNT Ssn, AVERAGE Salary}(\text{EMPLOYEE})$.

R

(a)

Dno	No_of_employees	Average_sal
5	4	33250
4	3	31000
1	1	55000

(b)

Dno	Count_ssn	Average_salary
5	4	33250
4	3	31000
1	1	55000

(c)

Count_ssn	Average_salary
8	35125

2.5.3 Recursive Closure Operations

Recursive closure operation is applied to a recursive relationship between tuples of the same type, such as the relationship between an employee and a supervisor.

Example : Retrieve all supervisees of an employee *e* at all levels—that is, all employees *e*' directly supervised by *e*, all employees *e*'ℑ directly supervised by each employee *e*', all employees *e*''' directly supervised by each employee *e*'' and so on.

$$\text{BORG\_SSN} \leftarrow \pi_{\text{Ssn}}(\sigma_{\text{Fname='James' AND Lname='Borg'}}(\text{EMPLOYEE}))$$
$$\text{SUPERVISION}(\text{Ssn1, Ssn2}) \leftarrow \pi_{\text{Ssn,Super\_ssn}}(\text{EMPLOYEE})$$
$$\text{RESULT1}(\text{Ssn}) \leftarrow \pi_{\text{Ssn1}}(\text{SUPERVISION} \bowtie_{\text{Ssn2=Ssn}} \text{BORG\_SSN})$$

SUPERVISION

(Borg's Ssn is 888665555)

(Ssn)	(Super_ssn)
Ssn1	Ssn2
123456789	333445555
333445555	888665555
999887777	987654321
987654321	888665555
666884444	333445555

RESULT1

Ssn
333445555
987654321

(Supervised by Borg)

987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Figure 1.2.3(b) : One possible database state for the COMPANY relational database schema.

A database state that does not obey all the integrity constraints is called **Invalid state** and a state that satisfies all the constraints in the defined set of integrity constraints IC is called a **Valid state**

hehejejjdjdndj

Attributes that represent the same real-world concept may or may not have identical names in different relations. For example, the Dnumber attribute in both DEPARTMENT and DEPT_LOCATIONS stands for the same real-world concept—the number given to a department. That same concept is called Dno in EMPLOYEE and Dnum in PROJECT.

Alternatively, attributes that represent different concepts may have the same name in different relations. For example, we could have used the attribute name Name for both Pname of PROJECT and Dname of DEPARTMENT; in this case, we would have two attributes that share the same name but represent different realworld concepts—project names and department names.

1.2.4 Integrity, Referential Integrity, and Foreign Keys

Entity integrity constraint

The entity integrity constraint states that no primary key value can be NULL. This is because the primary key value is used to identify individual tuples in a relation. Having NULL values for the primary key implies that we cannot identify some tuples. For example, if two or more tuples had NULL for their primary keys, we may not be able to distinguish them if we try to reference them from other relations.

Key constraints and entity integrity constraints are specified on individual relations.

Referential integrity constraint

The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

For example COMPANY database, the attribute Dno of EMPLOYEE gives the department number for which each employee works; hence, its value in every EMPLOYEE tuple must match the Dnumber value of some tuple in the DEPARTMENT relation.

To define referential integrity more formally, first we define the concept of a *foreign key*. The conditions for a foreign key, given below, specify a referential integrity constraint between the two relation schemas R_1 and R_2.

A set of attributes FK in relation schema R_1 is a **foreign key** of R_1 that **references** relation R_2 if it satisfies the following rules:

1. Attributes in FK have the same domain(s) as the primary key attributes PK of R_2; the attributes FK are said to **reference** or **refer to** the relation R_2.