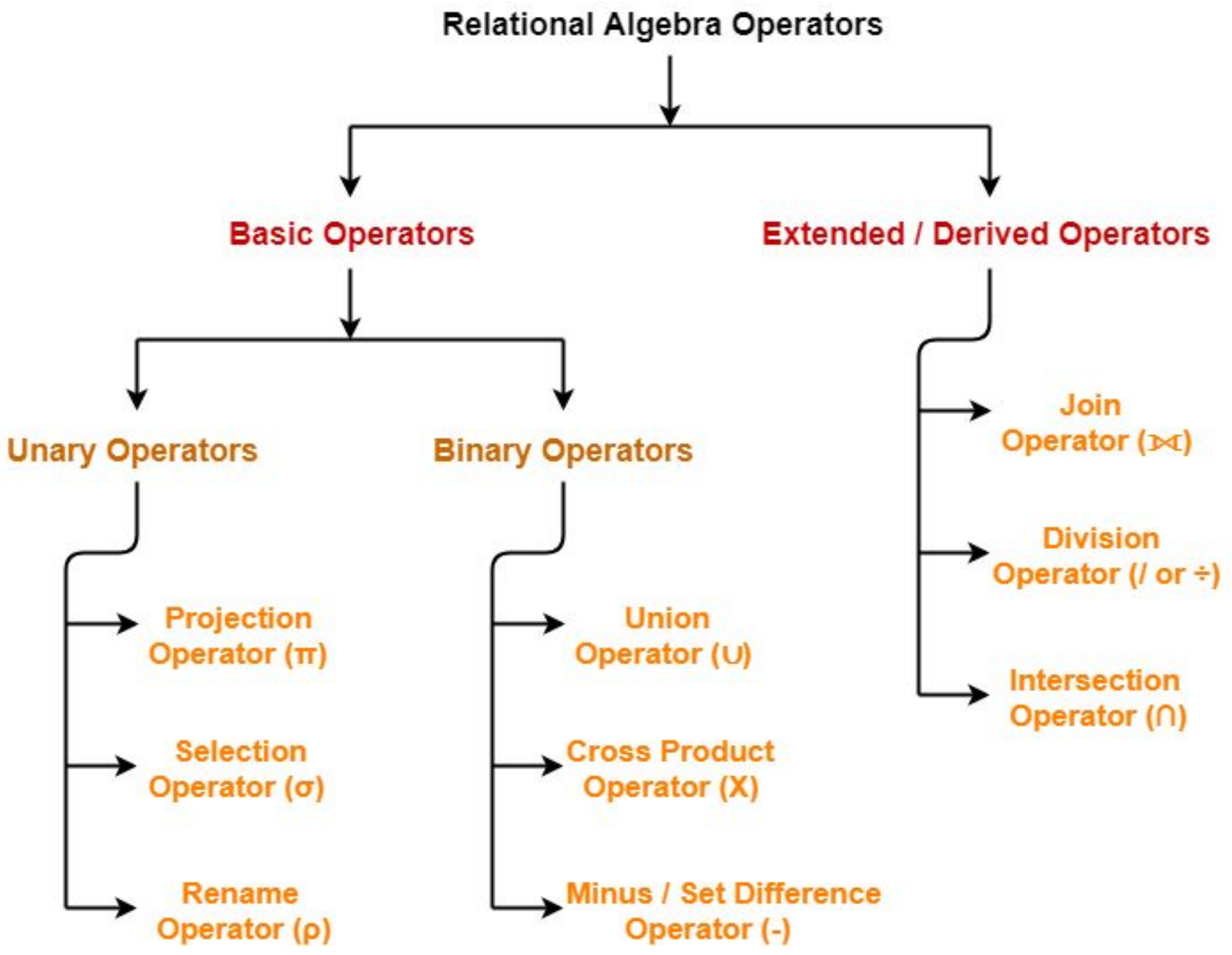
Databases 2022

Darko Bozhinoski,
Ph.D. in Computer Science

Agenda

- Relational Algebra (Recap)
- Relational Database Design by ER-to-Relational Mapping
- Introduction to SQL



Two types of operations:

- set operations from set theory

- operations developed specifically for relational databases

A Complete Set of Relational Algebra Operations

- Set of relational algebra operations {σ, π, ∪, ρ, –, ×} is a **complete set**
  - Any relational algebra operation can be expressed as a sequence of operations from this set
  - **INTERSECTION** can be expressed by using UNION and MINUS as follows: $R \cap S \equiv (R \cup S) - ((R - S) \cup (S - R))$
  - **JOIN** can be specified as a CARTESIAN PRODUCT followed by a **SELECT** operation: $R_{<condition>} S \equiv \sigma_{<condition>}(R \times S)$
  - **NATURAL JOIN can be specified as a CARTESIAN PRODUCT preceded by RENAME and followed by SELECT and PROJECT operations**

The DIVISION Operation

- Denoted by ÷
- Example: retrieve the names of employees who work on all the projects that 'John Smith' works on
- Apply to relations $R(Z) \div S(X)$
  - Attributes of R are a subset of the attributes of S

The DIVISION operation. (a) Dividing SSN_PNOS by SMITH_PNOS. (b) $T \leftarrow R \div S$.

(a)

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

(b)

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

S

A
a1
a2
a3

T

B
b1
b4

Example Division

Operations of Relational Algebra

Table 6.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<join condition>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<join condition>}} R_2$, OR $R_1 \bowtie_{(\text{<join attributes 1>}),\ (\text{<join attributes 2>})} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\text{<join condition>}} R_2$, OR $R_1 *_{(\text{<join attributes 1>}),\ (\text{<join attributes 2>})} R_2$ OR $R_1 * R_2$

Operations of Relational Algebra

Table 6.1 Operations of Relational Algebra

UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Tools to practise:

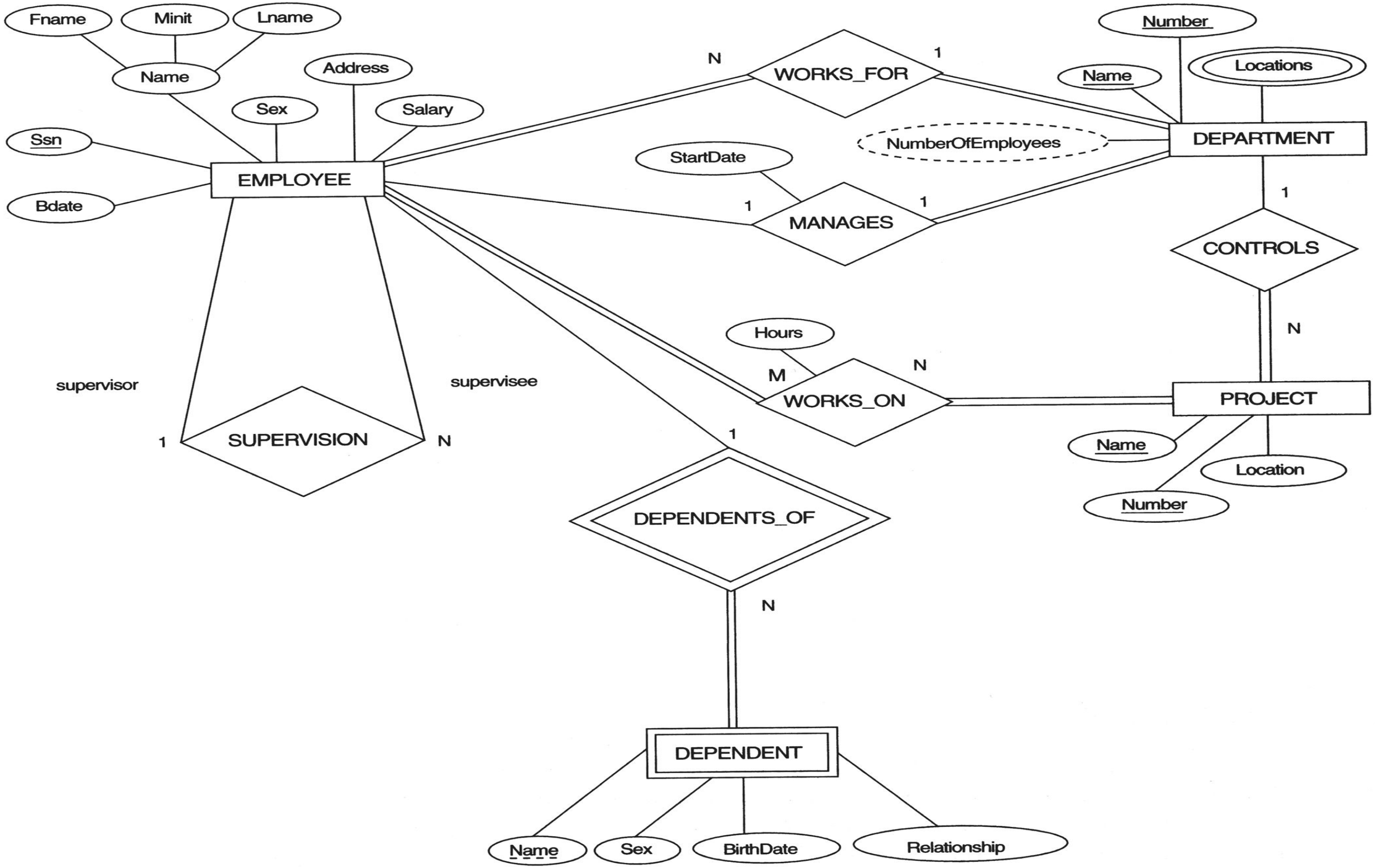
- https://dbis-uibk.github.io/relax/calc/local/uibk/local/4
- https://ltworf.github.io/relational/allowed_expressions.html

Relational Database Design by ER-to-Relational Mapping

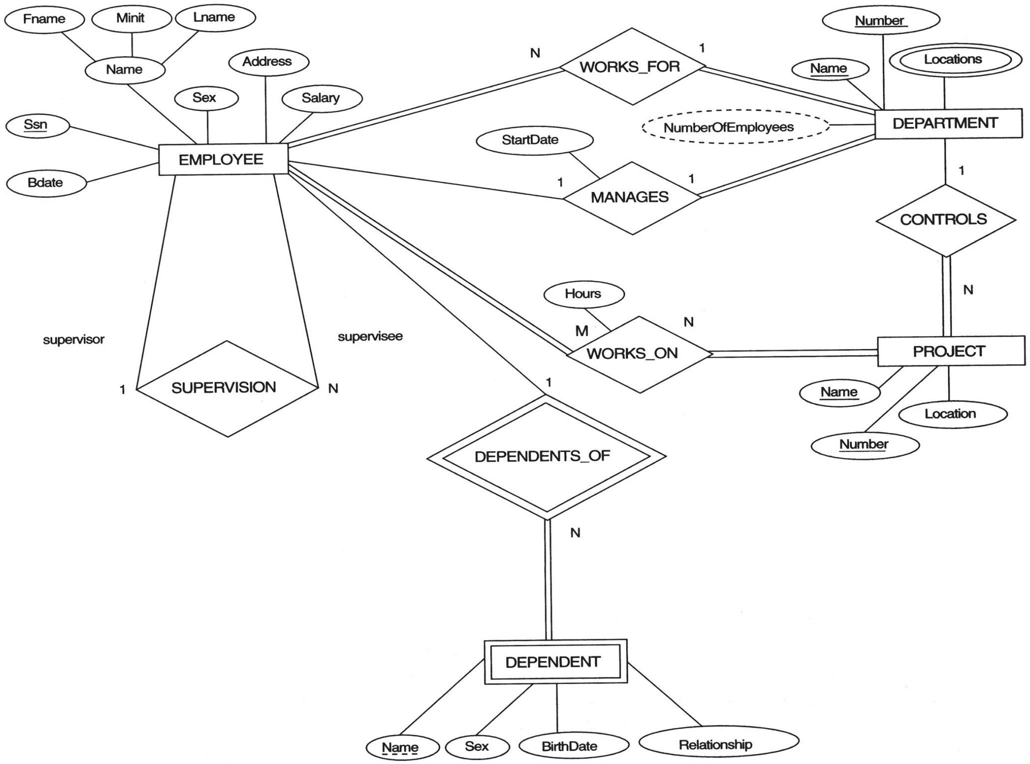
ER-to-Relational Mapping Algorithm

- **Step 1: Mapping of Regular Entity Types.**
  - For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E.
  - Choose one of the key attributes of E as the primary key for R.
  - If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R.

The ER conceptual schema diagram for the COMPANY database.

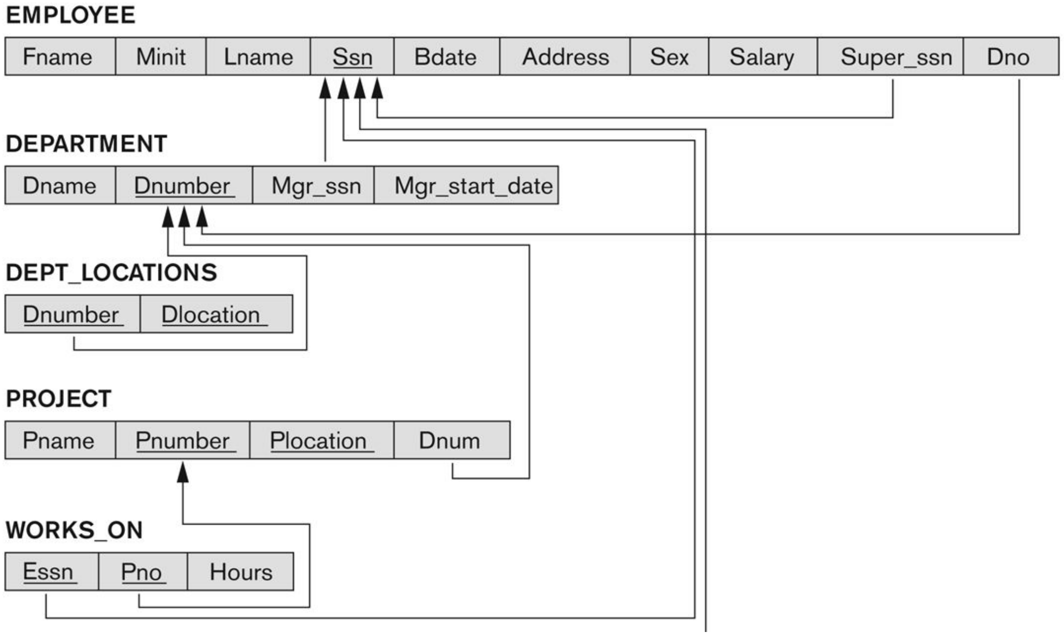


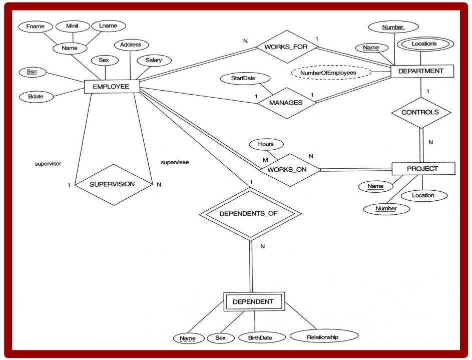
The ER conceptual schema diagram for the COMPANY database.



- **Example:** We create the relations EMPLOYEE, DEPARTMENT, and PROJECT in the relational schema corresponding to the regular entities in the ER diagram.
  - SSN, DNUMBER, and PNUMBER are the primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT as shown.

Partial Result of mapping the COMPANY ER schema into a relational schema.





ER-to-Relational Mapping Algorithm

- **Step 2: Mapping of Weak Entity Types**
  - For each weak entity type W in the ER schema with owner entity type E, create a relation R and include all simple attributes (or simple components of composite attributes) of W as attributes of R.
  - Also, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).
  - The primary key of R is the *combination of* the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any.
- **Example:** The relation DEPENDENT in this step corresponds to the weak entity type DEPENDENT.
  - Include the primary key SSN of the EMPLOYEE relation as a foreign key attribute of DEPENDENT (renamed to ESSN).
  - The primary key of the DEPENDENT relation is the combination {ESSN, DEPENDENT_NAME} because DEPENDENT_NAME is the partial key of DEPENDENT.

ER-to-Relational Mapping Algorithm

- **Step 3: Mapping of Binary 1:1 Relation 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:
  1. **Foreign Key approach:** Choose one of the relations-say S-and include 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.
     - Example: 1:1 relation MANAGES is mapped by choosing the participating entity type DEPARTMENT to serve in the role of S, because its participation in the MANAGES relationship type is total.
  2. **Merged relation option:** An alternate mapping of a 1:1 relationship type is possible by merging the two entity types and the relationship into a single relation. This may be appropriate when both participations are total.
  3. **Cross-reference or relationship relation option:** The third alternative is to 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.

ER-to-Relational Mapping Algorithm

- **Step 4: Mapping of Binary 1:N Relationship Types.**
  - For each regular binary 1:N relationship type R, identify the relation S that represent 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 of the 1:N relation type as attributes of S.
- Example: 1:N relationship types WORKS_FOR, CONTROLS, and SUPERVISION in the figure.
  - For WORKS_FOR we include the primary key DNUMBER of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it DNO.

ER-to-Relational Mapping Algorithm

- **Step 5: Mapping of Binary M:N Relationship Types.**
  - For each regular binary M:N relationship type R, *create a new relation* S to represent R.
  - Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; *their combination will form the primary key* of S.
  - Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of S.
- **Example:** The M:N relationship type WORKS_ON from the ER diagram is mapped by creating a relation WORKS_ON in the relational database schema.
  - The primary keys of the PROJECT and EMPLOYEE relations are included as foreign keys in WORKS_ON and renamed PNO and ESSN, respectively.
  - Attribute HOURS in WORKS_ON represents the HOURS attribute of the relation type. The primary key of the WORKS_ON relation is the combination of the foreign key attributes {ESSN, PNO}.

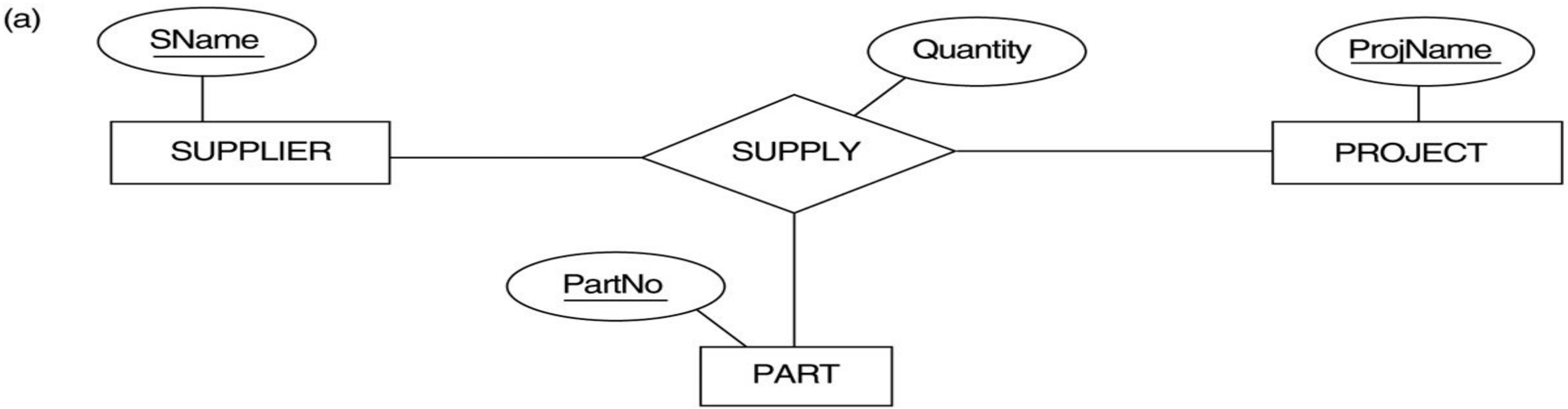
ER-to-Relational Mapping Algorithm

- **Step 6: Mapping of Multivalued attributes.**
  - For each multivalued attribute A, create a new relation R.
  - This relation R will include an attribute corresponding to A, plus the primary key attribute K-as a foreign key in R-of the relation that represents the entity type of relationship type that has A as an attribute.
  - The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components.
- **Example:** The relation DEPT_LOCATIONS is created.
  - 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 R is the combination of {DNUMBER, DLOCATION}.

ER-to-Relational Mapping Algorithm

- **Step 7: Mapping of N-ary Relationship Types.**
  - For each n-ary relationship type R, where n>2, create a new relationship S to represent R.
  - Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types.
  - Also include any simple attributes of the n-ary relationship type (or simple components of composite attributes) as attributes of S.

Exercise.



Map the following ER Diagram to Relational model.

SUPPLIER

SNAME	. . .

PROJECT

PROJNAME	. . .

PART

PARTNO	. . .

SUPPLY

SNAME	PROJNAME	PARTNO	QUANTITY

Summary of Mapping constructs and constraints

Correspondence between ER and Relational Models

ER Model	Relational Model
Entity type	“Entity” relation
1:1 or 1:N relationship type	Foreign key (or “relationship” relation)
M:N relationship type	“Relationship” relation and two foreign keys
n-ary relationship type	“Relationship” relation and n foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

Introduction to SQL

SQL OVERVIEW

- ❖ SQL: Structured Query Language
- ❖ The standard for relational database management systems (RDBMS)
- ❖ RDBMS: A database management system that manages data as a collection of tables in which all relationships are represented by common values in related tables

HISTORY OF SQL

- 1970–E. F. Codd develops relational database concept
- 1974-1979–System R with Sequel (later SQL) created at IBM Research Lab
- 1979–Oracle markets first relational DB with SQL
- 1981 – SQL/DS first available RDBMS system on DOS/VSE
- 1986– SQL standard released
- 1989, 1992, 1999, 2003, 2006, 2008–Major standard updates
- Current–SQL is supported by most major database vendors

PURPOSE OF SQL STANDARD

- ❖ Specify syntax/semantics for data definition and manipulation
- ❖ Define data structures and basic operations
- ❖ Enable portability of database definition and application modules
- ❖ Allow for later growth/enhancement to standard (referential integrity, transaction management, user-defined functions, extended join operations, etc.)

BENEFITS OF A STANDARDIZED RELATIONAL LANGUAGE

- ❖ Reduced training costs
- ❖ Productivity
- ❖ Application portability
- ❖ Application longevity
- ❖ Reduced dependence on a single vendor
- ❖ Cross-system communication

SQL ENVIRONMENT

- **Catalog**
  - A set of schemas that constitute the description of a database
- **Schema**
  - The structure that contains descriptions of objects created by a user (base tables, views, constraints)
- **Data Definition Language (DDL)**
  - Commands that define a database, including creating, altering, and dropping tables and establishing constraints (CREATE< ALTER< DROP_
- **Data Manipulation Language (DML)**
  - Commands that maintain and query a database (INSERT, UPDATE, DELETE, SELECT)
- **Data Control Language (DCL)**
  - Commands that control a database, including administering privileges and committing data (GRANT, REVOKE)

SQL DATA TYPES

TABLE 6-2 Sample SQL Data Types

String	CHARACTER (CHAR)	Stores string values containing any characters in a character set. CHAR is defined to be a fixed length.
	CHARACTER VARYING (VARCHAR or VARCHAR2)	Stores string values containing any characters in a character set but of definable variable length.
	BINARY LARGE OBJECT (BLOB)	Stores binary string values in hexadecimal format. BLOB is defined to be a variable length. (Oracle also has CLOB and NCLOB, as well as BFILE for storing unstructured data outside the database.)
Number	NUMERIC	Stores exact numbers with a defined precision and scale.
	INTEGER (INT)	Stores exact numbers with a predefined precision and scale of zero.
Temporal	TIMESTAMP TIMESTAMP WITH LOCAL TIME ZONE	Stores a moment an event occurs, using a definable fraction-of-a-second precision.Value adjusted to the user's session time zone (available in Oracle and MySQL)
Boolean	BOOLEAN	Stores truth values: TRUE, FALSE, or UNKNOWN.

SQL DATABASE DEFINITION

- Data Definition Language (DDL)
- Major CREATE statements:
  - **CREATE SCHEMA**–defines a portion of the database owned by a particular user
  - **CREATE TABLE**–defines a new table and its columns
  - **CREATE VIEW**–defines a logical table from one or more tables or views

STEPS IN TABLE CREATION

- ❖ Identify data types for attributes
- ❖ Identify columns that can and cannot be null
- ❖ Identify columns that must be unique (candidate keys)
- ❖ Identify primary key–foreign key mates
- ❖ Determine default values
- ❖ Identify constraints on columns (domain specifications)
- ❖ Create the table and associated indexes

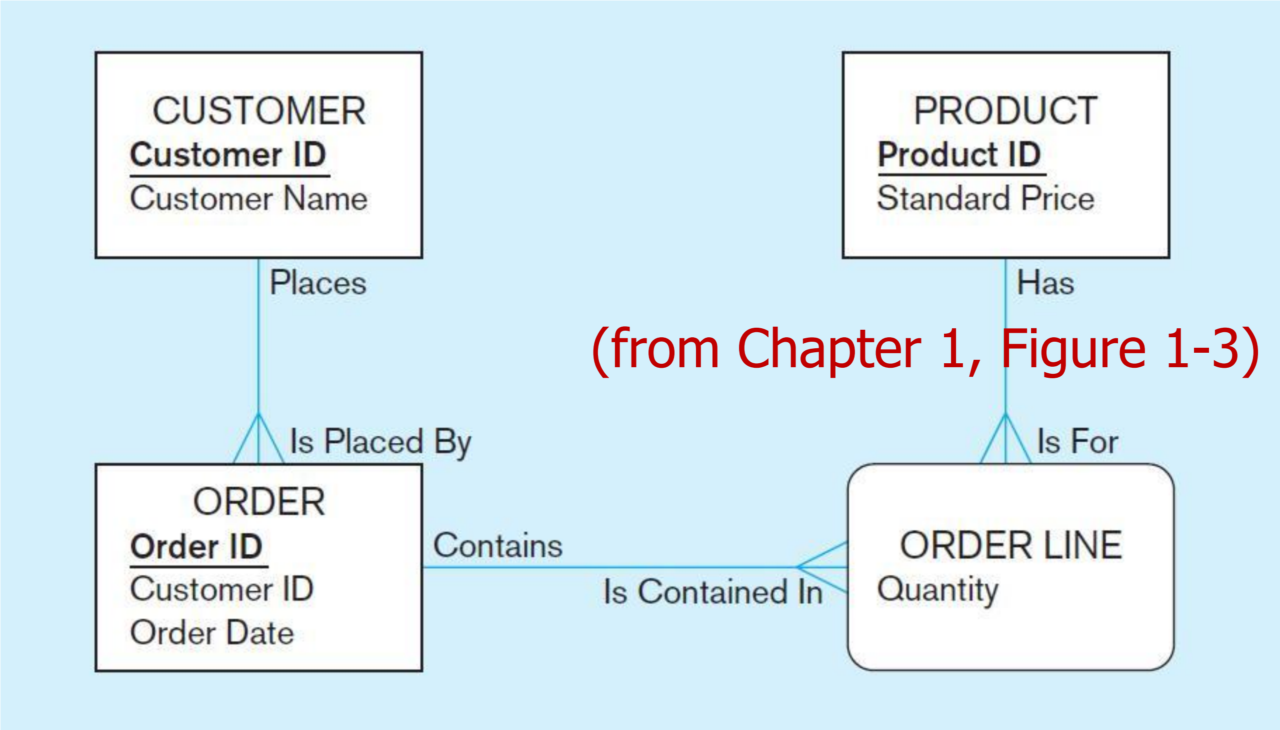
General syntax for CREATE TABLE statement used in data definition language

CREATE TABLE *tablename*
({*column definition*   [*table constraint*] } . , . .
[ON COMMIT {DELETE | PRESERVE} ROWS]);

where *column definition* ::=
column_name
  {*domain name* | datatype [(*size*)] }
  [*column_constraint_clause* . . .]
  [*default value*]
  [*collate clause*]

and *table constraint* ::=
  [CONSTRAINT *constraint_name*]
  Constraint_type [*constraint_attributes*]

CREATE TABLES FOR THIS ENTERPRISE DATA MODEL



(from Chapter 1, Figure 1-3)

SQL database definition commands

```
CREATE TABLE Customer_T
        (CustomerID              NUMBER(11,0)      NOT NULL,
         CustomerName            VARCHAR2(25)      NOT NULL,
         CustomerAddress         VARCHAR2(30),
         CustomerCity            VARCHAR2(20),
         CustomerState           CHAR(2),
         CustomerPostalCode      VARCHAR2(9),
CONSTRAINT Customer_PK PRIMARY KEY (CustomerID));

CREATE TABLE Order_T
        (OrderID                 NUMBER(11,0)      NOT NULL,
         OrderDate               DATE DEFAULT SYSDATE,
         CustomerID              NUMBER(11,0),
CONSTRAINT Order_PK PRIMARY KEY (OrderID),
CONSTRAINT Order_FK FOREIGN KEY (CustomerID) REFERENCES Customer_T(CustomerID));

CREATE TABLE Product_T
        (ProductID               NUMBER(11,0)      NOT NULL,
         ProductDescription      VARCHAR2(50),
         ProductFinish           VARCHAR2(20)
                   CHECK (ProductFinish IN ('Cherry', 'Natural Ash', 'White Ash',
                          'Red Oak', 'Natural Oak', 'Walnut')),
         ProductStandardPrice    DECIMAL(6,2),
         ProductLineID           INTEGER,
CONSTRAINT Product_PK PRIMARY KEY (ProductID));

CREATE TABLE OrderLine_T
        (OrderID                 NUMBER(11,0)      NOT NULL,
         ProductID               INTEGER           NOT NULL,
         OrderedQuantity         NUMBER(11,0),
CONSTRAINT OrderLine_PK PRIMARY KEY (OrderID, ProductID),
CONSTRAINT OrderLine_FK1 FOREIGN KEY (OrderID) REFERENCES Order_T(OrderID),
CONSTRAINT OrderLine_FK2 FOREIGN KEY (ProductID) REFERENCES Product_T(ProductID));
```

Overall table definitions

Defining attributes and their data types

```
CREATE TABLE Product_T
        (ProductID                  NUMBER(11,0)      NOT NULL,
         ProductDescription         VARCHAR2(50),
         ProductFinish              VARCHAR2(20)
                    CHECK (ProductFinish IN ('Cherry', 'Natural Ash', 'White Ash',
                           'Red Oak', 'Natural Oak', 'Walnut')),
         ProductStandardPrice       DECIMAL(6,2),
         ProductLineID              INTEGER,
CONSTRAINT Product_PK PRIMARY KEY (ProductID));
```

Non-nullable specification

```
CREATE TABLE Product_T
        (ProductID                NUMBER(11,0)     NOT NULL,
         ProductDescription       VARCHAR2(50),
         ProductFinish            VARCHAR2(20)
                  CHECK (ProductFinish IN ('Cherry', 'Natural Ash', 'White Ash',
                              'Red Oak', 'Natural Oak', 'Walnut')),
         ProductStandardPrice     DECIMAL(6,2),
         ProductLineID            INTEGER,
CONSTRAINT Product_PK PRIMARY KEY (ProductID));
```

Primary keys can never have NULL values

Identifying primary key

Non-nullable specifications

```
CREATE TABLE OrderLine_T
        (OrderID                NUMBER(11,0)    NOT NULL,
         ProductID              INTEGER         NOT NULL,
         OrderedQuantity        NUMBER(11,0),
CONSTRAINT OrderLine_PK PRIMARY KEY (OrderID, ProductID),
CONSTRAINT OrderLine_FK1 FOREIGN KEY (OrderID) REFERENCES Order_T(OrderID),
CONSTRAINT OrderLine_FK2 FOREIGN KEY (ProductID) REFERENCES Product_T(ProductID));
```

Primary key

Some primary keys are composite– composed of multiple attributes

Controlling the values in attributes

```
CREATE TABLE Order_T
        (OrderID                NUMBER(11,0)        NOT NULL,
         OrderDate              DATE DEFAULT SYSDATE,
         CustomerID             NUMBER(11,0),
CONSTRAINT Order_PK PRIMARY KEY (OrderID),
CONSTRAINT Order_FK FOREIGN KEY (CustomerID) REFERENCES Customer_T(CustomerID));

CREATE TABLE Product_T
        (ProductID              NUMBER(11,0)        NOT NULL,
         ProductDescription     VARCHAR2(50),
         ProductFinish          VARCHAR2(20)
                  CHECK (ProductFinish IN ('Cherry', 'Natural Ash', 'White Ash',
                                           'Red Oak', 'Natural Oak', 'Walnut')),
         ProductStandardPrice   DECIMAL(6,2),
         ProductLineID          INTEGER,
CONSTRAINT Product_PK PRIMARY KEY (ProductID));
```

Default value

Domain constraint

Identifying foreign keys and establishing relationships

```
CREATE TABLE Customer_T
        (CustomerID              NUMBER(11,0)        NOT NULL,
         CustomerName            VARCHAR2(25)        NOT NULL,
         CustomerAddress         VARCHAR2(30),
         CustomerCity            VARCHAR2(20),
         CustomerState           CHAR(2),
         CustomerPostalCode      VARCHAR2(9),
CONSTRAINT Customer_PK PRIMARY KEY (CustomerID));
```

Primary key of parent table

```
CREATE TABLE Order_T
        (OrderID                 NUMBER(11,0)        NOT NULL,
         OrderDate               DATE DEFAULT SYSDATE,
         CustomerID              NUMBER(11,0),
CONSTRAINT Order_PK PRIMARY KEY (OrderID),
CONSTRAINT Order_FK FOREIGN KEY (CustomerID) REFERENCES Customer_T(CustomerID));
```

Foreign key of dependent table

DATA INTEGRITY CONTROLS

- ❖ Referential integrity–constraint that ensures that foreign key values of a table must match primary key values of a related table in 1:M relationships
- ❖ Restricting:
  - ➢ Deletes of primary records
  - ➢ Updates of primary records
  - ➢ Inserts of dependent records

CHANGING TABLES

- **ALTER TABLE** statement allows you to change column specifications:

```
ALTER TABLE table_name alter_table_action;
```

- **Table Actions:**

```
ADD [COLUMN] column_definition
ALTER [COLUMN] column_name SET DEFAULT default-value
ALTER [COLUMN] column_name DROP DEFAULT
DROP [COLUMN] column_name [RESTRICT] [CASCADE]
ADD table_constraint
```

- Example (adding a new column with a default value):

```
ALTER TABLE CUSTOMER_T
ADD COLUMN CustomerType VARCHAR2 (2) DEFAULT "Commercial";
```

REMOVING TABLES

DROP TABLE statement allows you to remove tables from your schema:

- **DROP TABLE** CUSTOMER

INSERT STATEMENT

- Adds one or more rows to a table
- Inserting into a table

```
INSERT INTO Customer_T VALUES
(001, 'Contemporary Casuals', '1355 S. Himes Blvd.', 'Gainesville', 'FL', 32601);
```

- Inserting a record that has some null attributes requires identifying the fields that actually get data

```
INSERT INTO Product_T (ProductID,
ProductDescription, ProductFinish, ProductStandardPrice)
    VALUES (1, 'End Table', 'Cherry', 175, 8);
```

- Inserting from another table

```
INSERT INTO CaCustomer_T
SELECT * FROM Customer_T
    WHERE CustomerState = 'CA';
```

CREATING TABLES WITH IDENTITY COLUMNS

```
CREATE TABLE Customer_T
(CustomerID INTEGER GENERATED ALWAYS AS IDENTITY
        (START WITH 1
        INCREMENT BY 1
        MINVALUE 1
        MAXVALUE 10000
        NO CYCLE),
CustomerName              VARCHAR2(25) NOT NULL,
CustomerAddress           VARCHAR2(30),
CustomerCity              VARCHAR2(20),
CustomerState             CHAR(2),
CustomerPostalCode        VARCHAR2(9),
CONSTRAINT Customer_PK PRIMARY KEY (CustomerID);
```

Introduced with SQL:2008

Inserting into a table does not require explicit customer ID entry or field list

INSERT INTO CUSTOMER_T VALUES (‘Contemporary Casuals’, ‘1355 S. Himes Blvd.’, ‘Gainesville’, ‘FL’, 32601);

DELETE STATEMENT

- Removes rows from a table
- Delete certain rows
  - *DELETE FROM CUSTOMER_T WHERE CUSTOMERCOUNTRY = ‘RU’;*
- Delete all rows
  - *DELETE FROM CUSTOMER_T;*

UPDATE STATEMENT

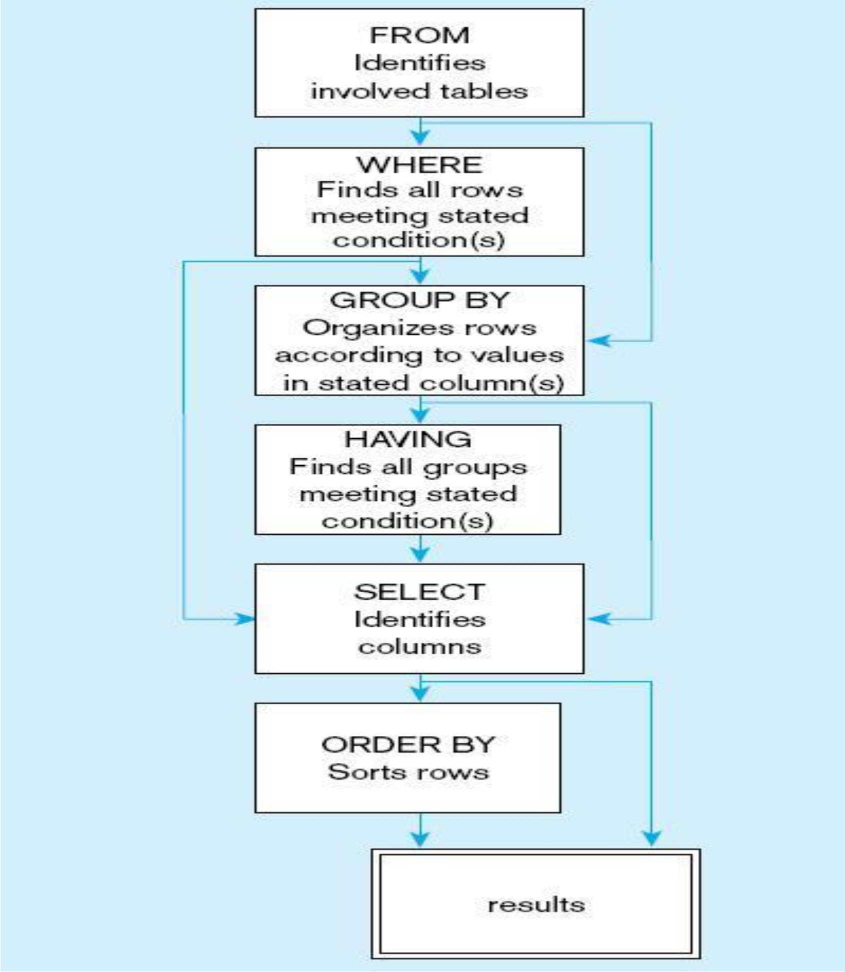
- Modifies data in existing rows

```
UPDATE Product_T
SET ProductStandardPrice = 775
    WHERE ProductID = 7;
```

SQL Retrievals

SELECT STATEMENT

- Used for queries on single or multiple tables
- Clauses of the SELECT statement:
  - **SELECT**
    - List the columns (and expressions) to be returned from the query
  - **FROM**
    - Indicate the table(s) or view(s) from which data will be obtained
  - **WHERE**
    - Indicate the conditions under which a row will be included in the result
  - **GROUP BY**
    - Indicate categorization of results
  - **HAVING**
    - Indicate the conditions under which a category (group) will be included
  - **ORDER BY**
    - Sorts the result according to specified criteria


FROM
Identifies
involved tables
WHERE
Finds all rows
meeting stated
condition(s)
GROUP BY
Organizes rows
according to values
in stated column(s)
HAVING
Finds all groups
meeting stated
condition(s)
SELECT
Identifies
columns
ORDER BY
Sorts rows
results

SELECT EXAMPLE

- Find products with standard price less than $275

```
SELECT ProductDescription, ProductStandardPrice
  FROM Product_T
    WHERE ProductStandardPrice < 275;
```

TABLE 6-3 Comparison Operators in SQL

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
!=	Not equal to

SELECT EXAMPLE USING ALIAS

Alias is an alternative column or table name

```
SELECT CUST.CustomerName AS Name, CUST.CustomerAddress
  FROM ownerid.Customer_T AS Cust
    WHERE Name = 'Home Furnishings';
```

Here, *CUST* is a table alias and *Name* is a column alias

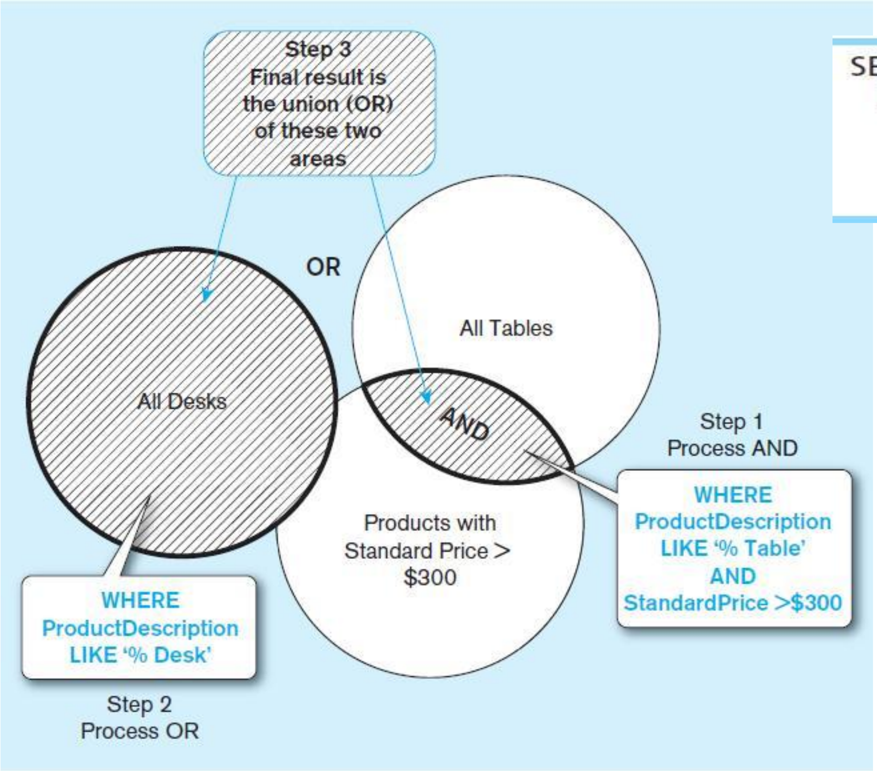
SELECT EXAMPLE–BOOLEAN OPERATORS

AND, OR, and NOT Operators for customizing conditions in WHERE clause

```
SELECT ProductDescription, ProductFinish, ProductStandardPrice
  FROM Product_T
    WHERE ProductDescription LIKE '%Desk'
      OR ProductDescription LIKE '%Table'
      AND ProductStandardPrice > 300;
```

Note: the LIKE operator allows you to compare strings using wildcards. For example, the % wildcard in '%Desk' indicates that all strings that have any number of characters preceding the word "Desk" will be allowed.

Query without use of parentheses




```
SELECT ProductDescription, ProductFinish, ProductStandardPrice
  FROM Product_T
    WHERE ProductDescription LIKE '%Desk'
      OR ProductDescription LIKE '%Table'
      AND ProductStandardPrice > 300;
```

By default, processing order of Boolean operators is NOT, then AND, then OR

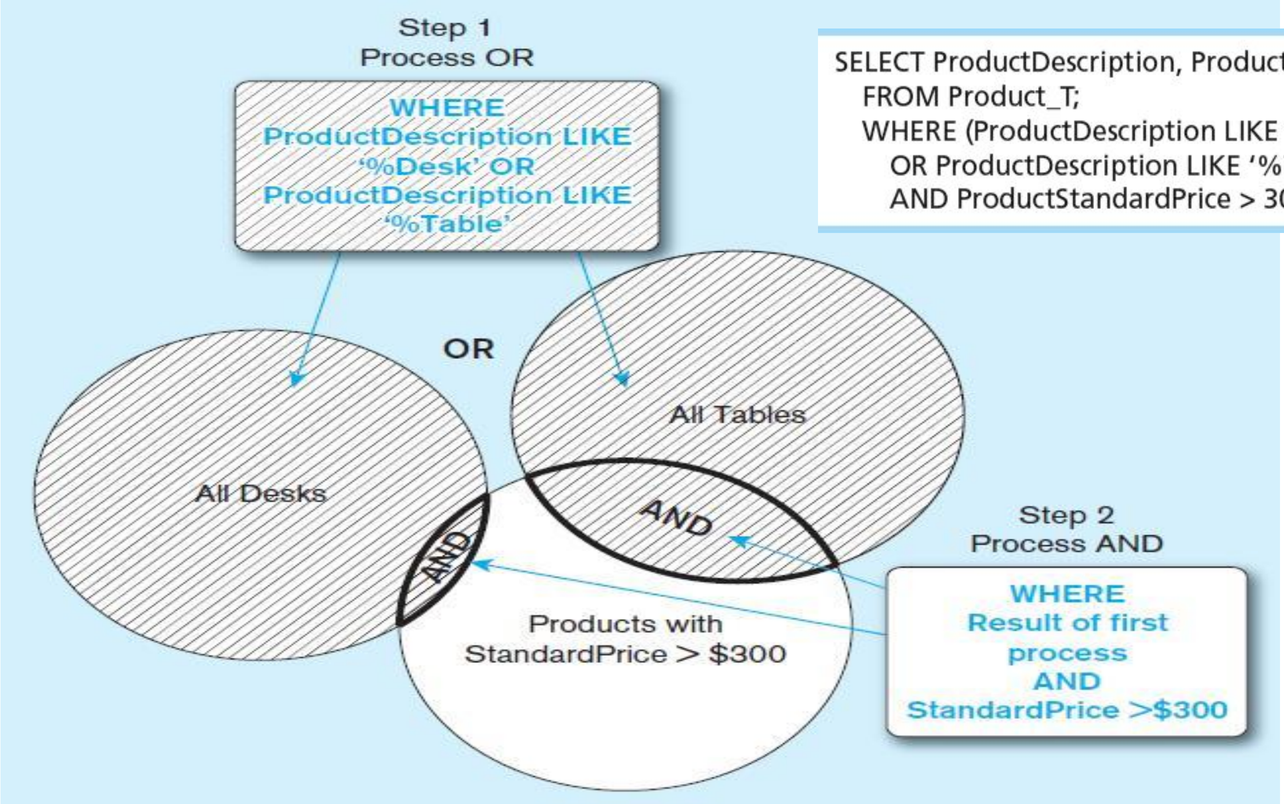
SELECT EXAMPLE–BOOLEAN OPERATORS

- With parentheses…these override the normal precedence of Boolean operators

```
SELECT ProductDescription, ProductFinish, ProductStandardPrice
   FROM Product_T;
   WHERE (ProductDescription LIKE '%Desk'
      OR ProductDescription LIKE '%Table')
      AND ProductStandardPrice > 300;
```

With parentheses, you can override normal precedence rules. In this case parentheses make the OR take place before the AND.

Boolean query B with use of parentheses




```
SELECT ProductDescription, ProductFinish, ProductStandardPrice
   FROM Product_T;
   WHERE (ProductDescription LIKE '%Desk'
      OR ProductDescription LIKE '%Table')
      AND ProductStandardPrice > 300;
```

SORTING RESULTS WITH ORDER BY CLAUSE

- Sort the results first by STATE, and within a state by the CUSTOMER NAME

```
SELECT CustomerName, CustomerCity, CustomerState
  FROM Customer_T
    WHERE CustomerState IN ('FL', 'TX', 'CA', 'HI')
      ORDER BY CustomerState, CustomerName;
```

Note: the IN operator in this example allows you to include rows whose CustomerState value is either FL, TX, CA, or HI. It is more efficient than separate OR conditions.

CATEGORIZING RESULTS USING GROUP BY CLAUSE

- For use with aggregate functions
  - ***Scalar aggregate***: single value returned from SQL query with aggregate function
  - ***Vector aggregate***: multiple values returned from SQL query with aggregate function (via GROUP BY)
  -

```
SELECT CustomerState, COUNT (CustomerState)
  FROM Customer_T
    GROUP BY CustomerState;
```

You can use single-value fields with aggregate functions if they are included in the GROUP BY clause.

QUALIFYING RESULTS BY CATEGORIES USING THE HAVING CLAUSE

- For use with **GROUP BY**

```
SELECT CustomerState, COUNT (CustomerState)
   FROM Customer_T
      GROUP BY CustomerState
      HAVING COUNT (CustomerState) > 1;
```

Like a **WHERE** clause, but it operates on groups (categories), not on individual rows. Here, only those groups with total numbers greater than 1 will be included in final result.

Reading Material

- Fundamentals of Database Systems. Ramez Elmasri and Shamkant B. Navathe. Pearson. **Chapter 9. and Chapter 5.**
- SQL Tutorial: **https://www.w3schools.com/sql/default.asp**


Q & A