Introduction to Database Systems

CSE 444

Lecture #6
Jan 22 2001

Announcements – I

- ⌘**Programming Assignment due on Thu (1/25)**

Using SQL in Applications

Reading: Section 7
(except 7.2, 7.4 – to be covered later)

Using SQL in Applications

- ⌘Business logic involves
  - ⧄Language Issues
    - ⊠Application code in a development language (Java, C++, Visual Basic)
  - ⧄Client-Server communication
    - ⊠Application connects and "does work" at database server

Language Issues

- ⌘Data Type issues (Mapping of Types)
- ⌘Reconcile Explicit iteration in Programming Language with set-oriented processing in SQL (Cursors)
- ⌘SQL generated on-the-fly (Dynamic SQL)

SQL Generated On-the-fly

- ⌘Static SQL without parameters:
  - ⧄Select * from Students
- ⌘Static SQL with parameters
  - ⧄Select * from students where Student_name = :sname
- ⌘Dynamic SQL
  - ⧄ An arbitrary string that represents a SQL statement
  - ⧄ Statement created at runtime

Processing SQL

- Key Steps
  - Parse SQL
  - Validate SQL against system catalog
  - Generate an "execution plan"
  - Optimize the execution plan
  - Execute the plan

Implication for SQL generated on-the-fly

- Static SQL
  - Execution plan may be generated at compilation time
- Static SQL with parameters
  - Almost as above
- Dynamic SQL
  - Compile time optimization not possible

Handling Dynamic SQL

- Runtime optimization
  - Compile only once at runtime
  - Execute multiple times
- Roughly:
  - Prepare statement_name from statement_variable
  - Execute statement_name using arg [, arg]

Client Server Communication

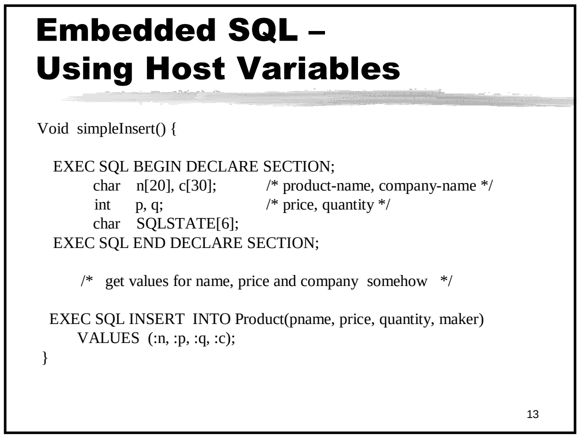
- Embedded SQL
- Call Level Interface

Embedded SQL

- Embed SQL statements in a host language program
  - Variables from the application program can be used in the SQL statement (host variables)
  - Processed by a SQL Preprocessor
  - Use cursors for multi-row output
  - Structure to return errors (SQLCA)

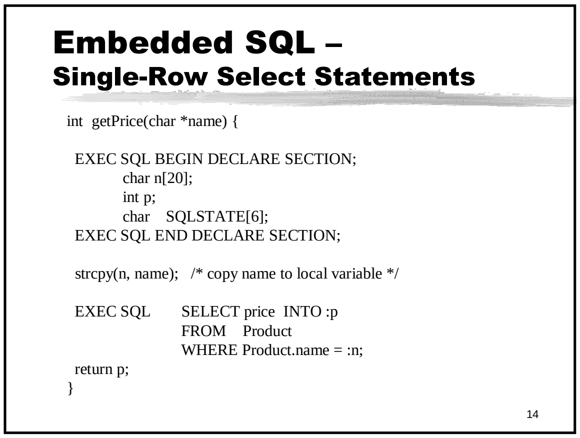
Compiling Embedded SQL

- Embedded SQL submitted to precompiler
  - One Precompiler/language supported by DBMS
- Precompiler produces 2 files
  - Source code + proprietary calls to DBMS routines
  - Database Request Module (all SQL statements)
- Next Steps
  - Source code => object file, Linker links object files + library routines
  - Binding utility generates executable SQL
- Execute!



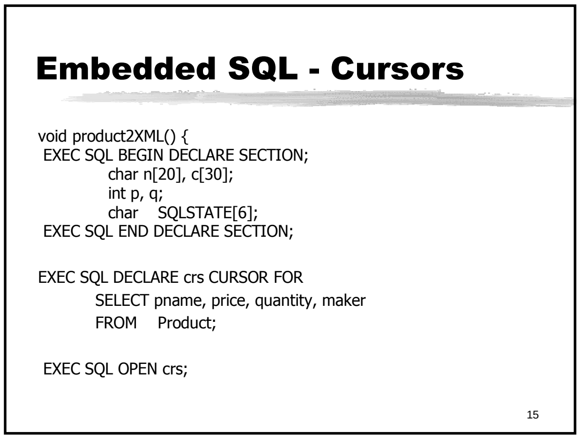
Embedded SQL – Using Host Variables

```
Void  simpleInsert() {

   EXEC SQL BEGIN DECLARE SECTION;
          char    n[20], c[30];          /* product-name, company-name */
          int     p, q;                  /* price, quantity */
          char    SQLSTATE[6];
   EXEC SQL END DECLARE SECTION;

        /*   get values for name, price and company  somehow   */

  EXEC SQL INSERT  INTO Product(pname, price, quantity, maker)
       VALUES  (:n, :p, :q, :c);
 }
```



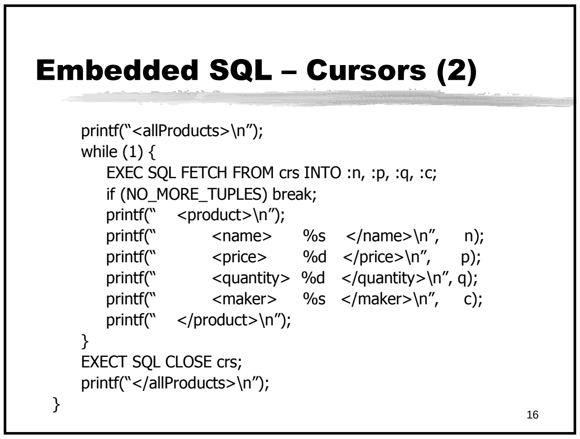
Embedded SQL – Single-Row Select Statements

```
int  getPrice(char *name) {

  EXEC SQL BEGIN DECLARE SECTION;
         char n[20];
         int p;
         char    SQLSTATE[6];
  EXEC SQL END DECLARE SECTION;

  strcpy(n, name);   /* copy name to local variable */

  EXEC SQL       SELECT price  INTO :p
                 FROM    Product
                 WHERE Product.name = :n;
  return p;
}
```



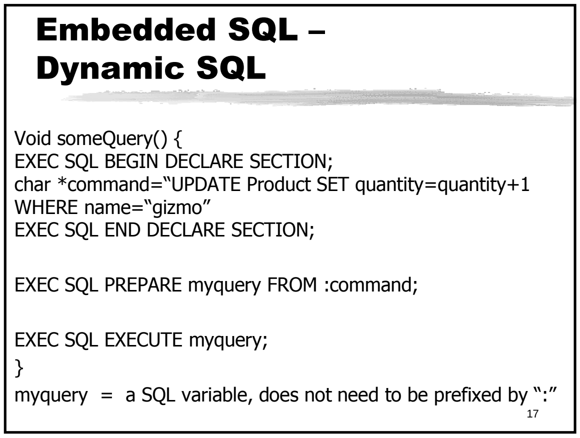
Embedded SQL - Cursors

```
void product2XML() {
 EXEC SQL BEGIN DECLARE SECTION;
           char n[20], c[30];
           int p, q;
           char    SQLSTATE[6];
 EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE crs CURSOR FOR
         SELECT pname, price, quantity, maker
         FROM    Product;

 EXEC SQL OPEN crs;
```



Embedded SQL – Cursors (2)

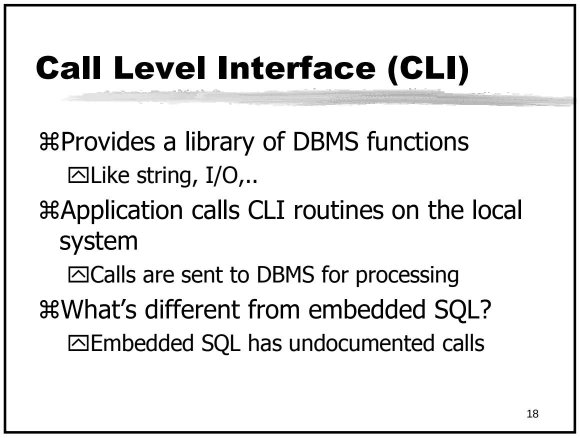
```
    printf("<allProducts>\n");
    while (1) {
        EXEC SQL FETCH FROM crs INTO :n, :p, :q, :c;
        if (NO_MORE_TUPLES) break;
        printf("    <product>\n");
        printf("          <name>      %s    </name>\n",     n);
        printf("          <price>      %d   </price>\n",     p);
        printf("          <quantity>  %d   </quantity>\n", q);
        printf("          <maker>     %s    </maker>\n",     c);
        printf("    </product>\n");
    }
    EXECT SQL CLOSE crs;
    printf("</allProducts>\n");
}
```



Embedded SQL – Dynamic SQL

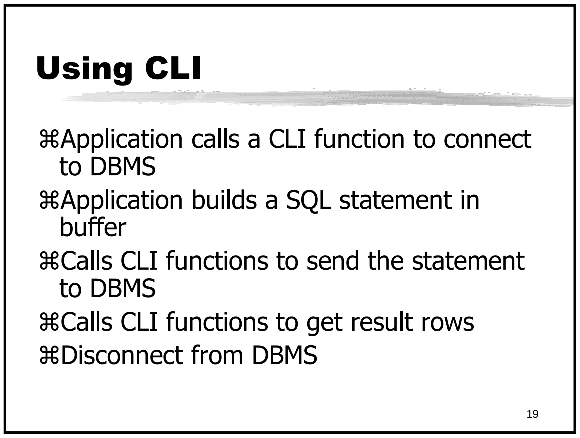
```
Void someQuery() {
EXEC SQL BEGIN DECLARE SECTION;
char *command="UPDATE Product SET quantity=quantity+1
WHERE name="gizmo"
EXEC SQL END DECLARE SECTION;

EXEC SQL PREPARE myquery FROM :command;

EXEC SQL EXECUTE myquery;
}
```

myquery = a SQL variable, does not need to be prefixed by ":"



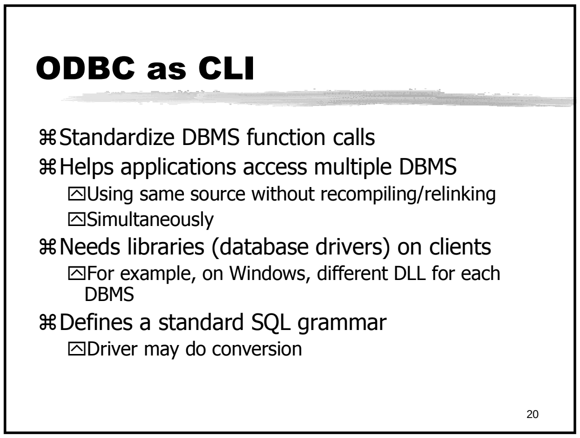
Call Level Interface (CLI)

- Provides a library of DBMS functions
  - Like string, I/O,..
- Application calls CLI routines on the local system
  - Calls are sent to DBMS for processing
- What's different from embedded SQL?
  - Embedded SQL has undocumented calls



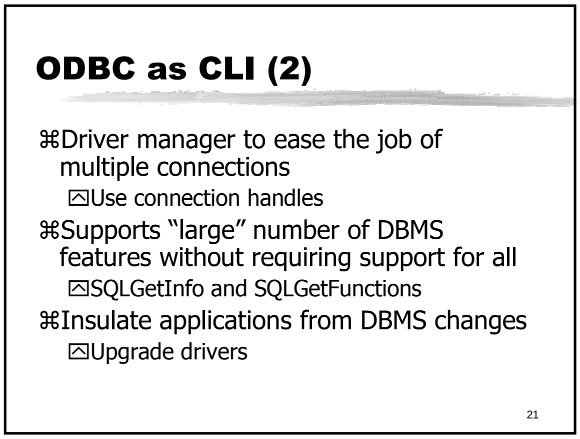
Using CLI

- Application calls a CLI function to connect to DBMS
- Application builds a SQL statement in buffer
- Calls CLI functions to send the statement to DBMS
- Calls CLI functions to get result rows
- Disconnect from DBMS



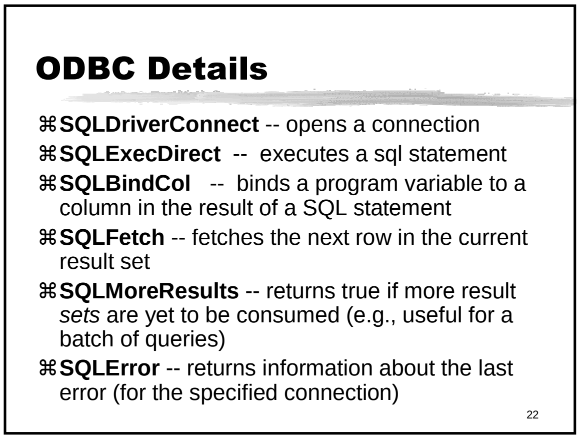
ODBC as CLI

- Standardize DBMS function calls
- Helps applications access multiple DBMS
  - Using same source without recompiling/relinking
  - Simultaneously
- Needs libraries (database drivers) on clients
  - For example, on Windows, different DLL for each DBMS
- Defines a standard SQL grammar
  - Driver may do conversion



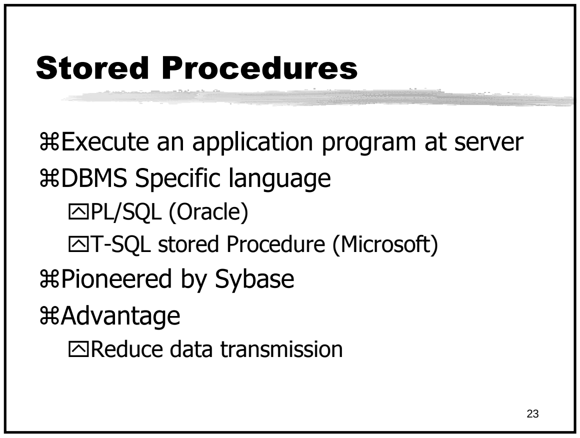
ODBC as CLI (2)

- Driver manager to ease the job of multiple connections
  - Use connection handles
- Supports "large" number of DBMS features without requiring support for all
  - SQLGetInfo and SQLGetFunctions
- Insulate applications from DBMS changes
  - Upgrade drivers



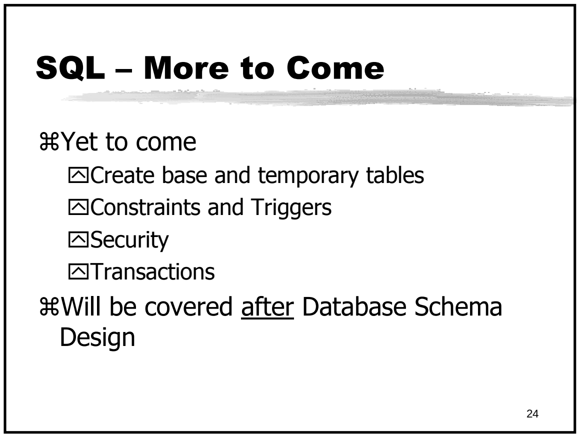
ODBC Details

- **SQLDriverConnect** -- opens a connection
- **SQLExecDirect** -- executes a sql statement
- **SQLBindCol** -- binds a program variable to a column in the result of a SQL statement
- **SQLFetch** -- fetches the next row in the current result set
- **SQLMoreResults** -- returns true if more result *sets* are yet to be consumed (e.g., useful for a batch of queries)
- **SQLError** -- returns information about the last error (for the specified connection)



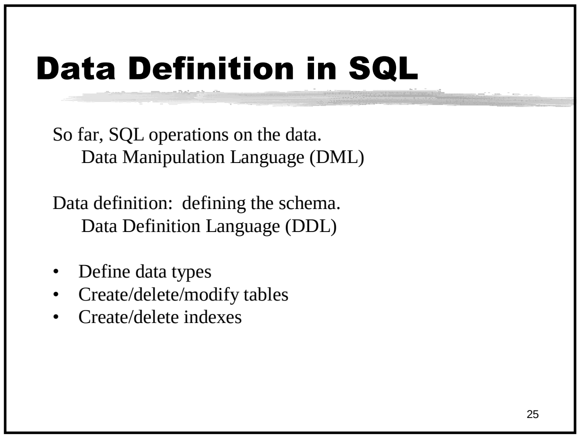
Stored Procedures

- Execute an application program at server
- DBMS Specific language
  - PL/SQL (Oracle)
  - T-SQL stored Procedure (Microsoft)
- Pioneered by Sybase
- Advantage
  - Reduce data transmission



SQL – More to Come

- Yet to come
  - Create base and temporary tables
  - Constraints and Triggers
  - Security
  - Transactions
- Will be covered <u>after</u> Database Schema Design

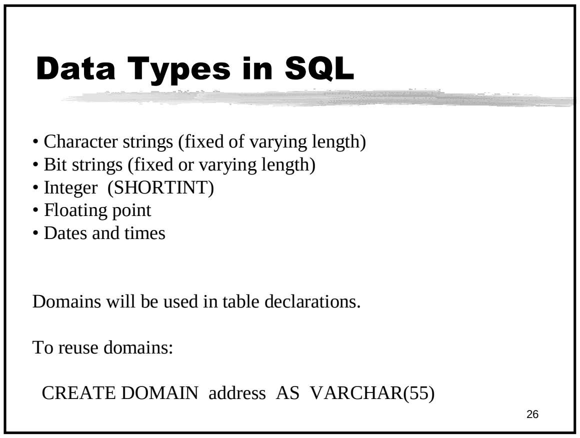


Data Definition in SQL

So far, SQL operations on the data.
    Data Manipulation Language (DML)

Data definition: defining the schema.
    Data Definition Language (DDL)

- Define data types
- Create/delete/modify tables
- Create/delete indexes



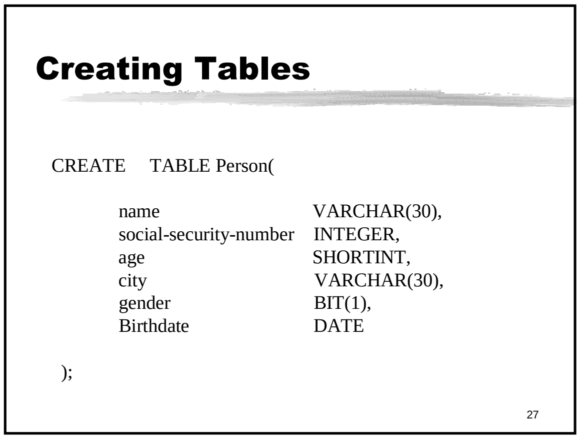
Data Types in SQL

- Character strings (fixed of varying length)
- Bit strings (fixed or varying length)
- Integer (SHORTINT)
- Floating point
- Dates and times

Domains will be used in table declarations.

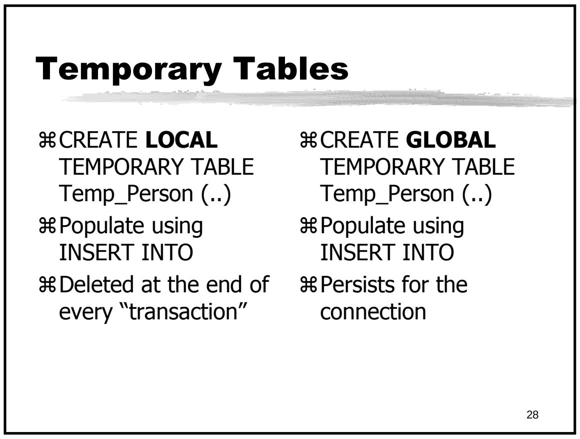
To reuse domains:

```
CREATE DOMAIN  address  AS  VARCHAR(55)
```



Creating Tables

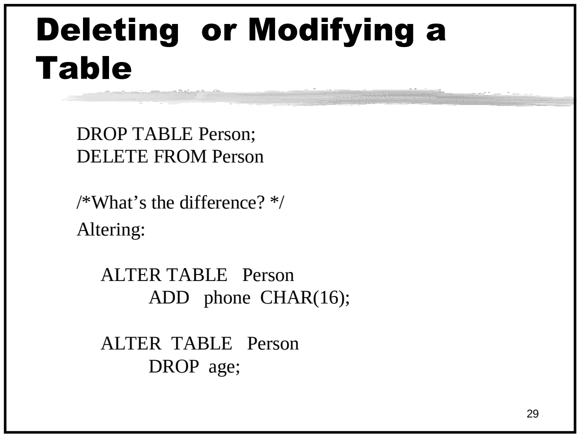
```
CREATE     TABLE Person(

        name                      VARCHAR(30),
        social-security-number    INTEGER,
        age                       SHORTINT,
        city                      VARCHAR(30),
        gender                    BIT(1),
        Birthdate                 DATE

 );
```



Temporary Tables

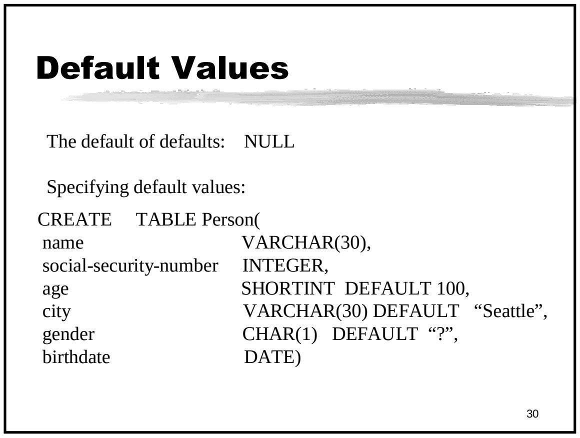
- ⌘CREATE **LOCAL** TEMPORARY TABLE Temp_Person (..)
- ⌘Populate using INSERT INTO
- ⌘Deleted at the end of every "transaction"

- ⌘CREATE **GLOBAL** TEMPORARY TABLE Temp_Person (..)
- ⌘Populate using INSERT INTO
- ⌘Persists for the connection



Deleting or Modifying a Table

```
DROP TABLE Person;
DELETE FROM Person

/*What's the difference? */
Altering:

   ALTER TABLE   Person
          ADD   phone  CHAR(16);

   ALTER  TABLE   Person
          DROP  age;
```



Default Values

The default of defaults: NULL

Specifying default values:

```
CREATE     TABLE Person(
 name                       VARCHAR(30),
 social-security-number     INTEGER,
 age                        SHORTINT  DEFAULT 100,
 city                       VARCHAR(30) DEFAULT   "Seattle",
 gender                     CHAR(1)   DEFAULT  "?",
 birthdate                  DATE)
```

Database Schema Design

Today's Reading:

Sec 2 (except 2.1 and ODL discussions) and
Sec 3.1- 3.4 (except 3.1)

Database Design

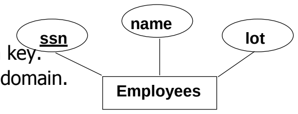
- Why do we need it?
  - Agree on structure of the database before deciding on a particular implementation.
- Consider issues such as:
  - What entities to model
  - How entities are related
  - What constraints exist in the domain
  - How to achieve *good* designs

Overview of Database Design

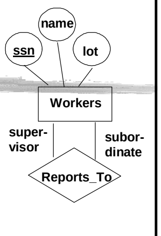
- Conceptual design: (ER Model is used at this stage.)
  - ER Diagram
    - What are the entities and relationships in the enterprise?
    - What are the integrity constraints or business rules that hold?
  - Map an ER diagram into a relational schema
- Schema Refinement (Normalization):
  - Check relational schema for redundancies and related anomalies.
- Physical Design:
  - Determine physical structures

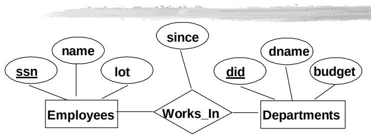
ER Model Basics

- Entity: Real-world object distinguishable from other objects. An entity is described (in DB) using a set of attributes.
- Entity Set: A collection of similar entities. E.g., all employees.
  - All entities in an entity set have the same set of attributes.
  - Each entity set has a key.
  - Each attribute has a domain.



ER Model Basics

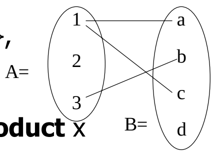




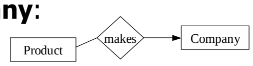
- Relationship: Association among two or more entities. E.g., Ed works in Pharmacy department.
  - Can have attributes to describe how entities are related
- Relationship Set: Collection of similar relationships.

What is a Relationship ?

- A mathematical definition:
  - if A, B are sets, then a relation R is a subset of A x B
- A={1,2,3}, B={a,b,c,d},
  R = {(1,a), (1,c), (3,b)}

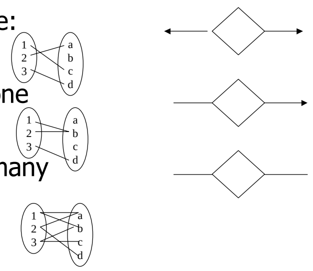


- **makes** is a subset of **Product** x **Company**:



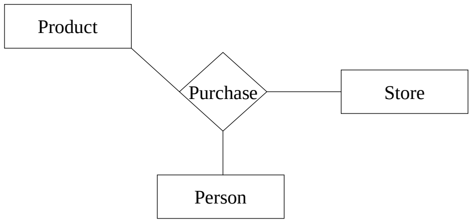
Multiplicity of E/R Relationships

- ⌘one-one:
- ⌘many-one
- ⌘many-many



Multi-way Relationships

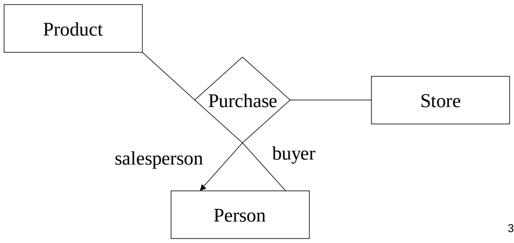
How do we model a purchase relationship between buyers, products and stores?



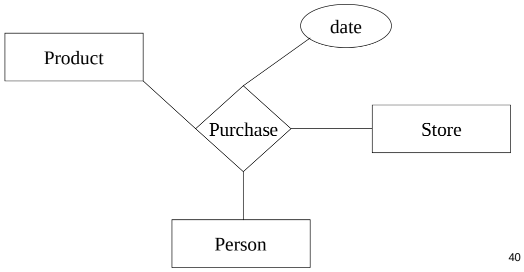
Can still model as a mathematical set (how ?)

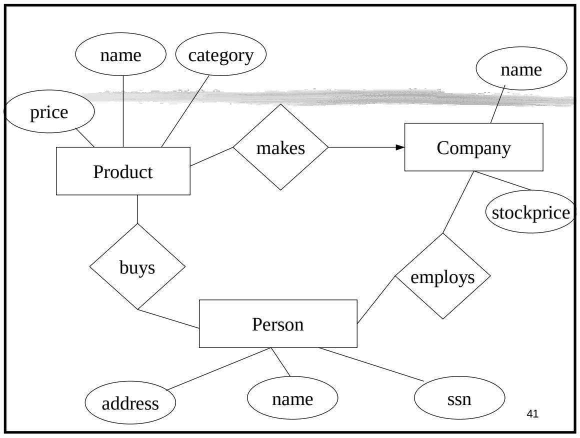
Roles in Relationships

What if we need an entity set twice in one relationship?

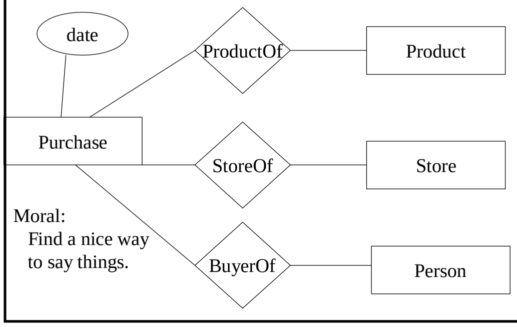


Attributes on Relationships





Converting Multi-way Relationships to Binary



Moral:
Find a nice way
to say things.

Recap: Conceptual Design

- *Conceptual design* follows requirements analysis:
  - Yields a high-level description of data to be stored
- ER model popular for conceptual design
  - Constructs are expressive, close to the way people think about their applications.
- Basic constructs: *entities*, *relationships*, and *attributes* (of entities and relationships).
- Note: There are many variations on ER model.

Recap: Conceptual Design Using the ER Model

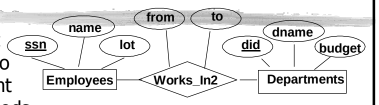
- Design choices:
  - Should a concept be modeled as an entity or an attribute?
  - Should a concept be modeled as an entity or a relationship?
  - Identifying relationships: Binary or ternary?

Design Choices: Entity vs. Attribute

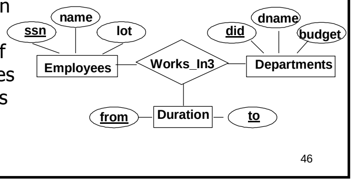
- Should address be an attribute of Employees or an entity (connected to Employees by a relationship)?
- Depends upon the use we want to make of address information, and the semantics of the data:
  - ☒If we have several addresses per employee, *address* must be an entity (since attributes cannot be set-valued).
  - ☒If the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city, *address* must be modeled as an entity (since attribute values are atomic).

Entity vs. Attribute (Contd.)

- Works_In2 does not allow an employee to work in a department for two or more periods.

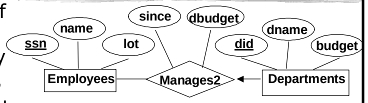


- Similar to the problem of wanting to record several addresses for an employee: we want to record several values of the descriptive attributes for each instance of this relationship.

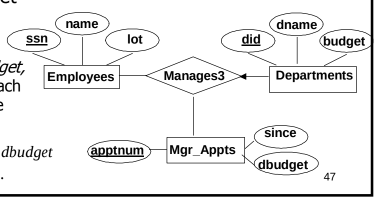


Design Choice Entity vs. Relationship

- First ER diagram OK if a manager gets a separate discretionary budget for each dept.



- What if a manager gets a discretionary budget that covers *all* managed depts?
  - Redundancy of *dbudget,* which is stored for each dept managed by the manager.



Misleading: suggests *dbudget* tied to managed dept.

Comments on ER Models

- ER design is *subjective*. There are often many ways to model a given scenario! Analyzing alternatives can be tricky, especially for a large enterprise. Common choices include:
  - Entity vs. attribute, entity vs. relationship, binary or n-ary relationship, roles, etc.
- Need to model constraints on data
  - To follow ..