Structured Query Language

MSBA 7024 / MACC 7020

Database Design and Management

Objectives

- Definition of terms
- Interpret history and role of SQL
- Define a database using SQL data definition language
- Write single table queries using SQL
- Establish referential integrity using SQL
- Write multiple table SQL queries
- Define and use different types of joins
- Write noncorrelated and correlated subqueries
- Establish transaction integrity in SQL
- Understand triggers and stored procedures

SQL Overview

- 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. 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
- 1986: ANSI SQL standard released
- 1989, 1992, 1999, 2003, 2006, 2008, 2011, 2016: Major ANSI standard updates
- Current: SQL is supported by all major relational database vendors

Purpose of SQL Standard

- Specify syntax/semantics for data definition and manipulation
- Define data structures
- Enable portability
- Specify minimal (level 1) and complete (level 2) standards
- Allow for later growth/enhancement to standard

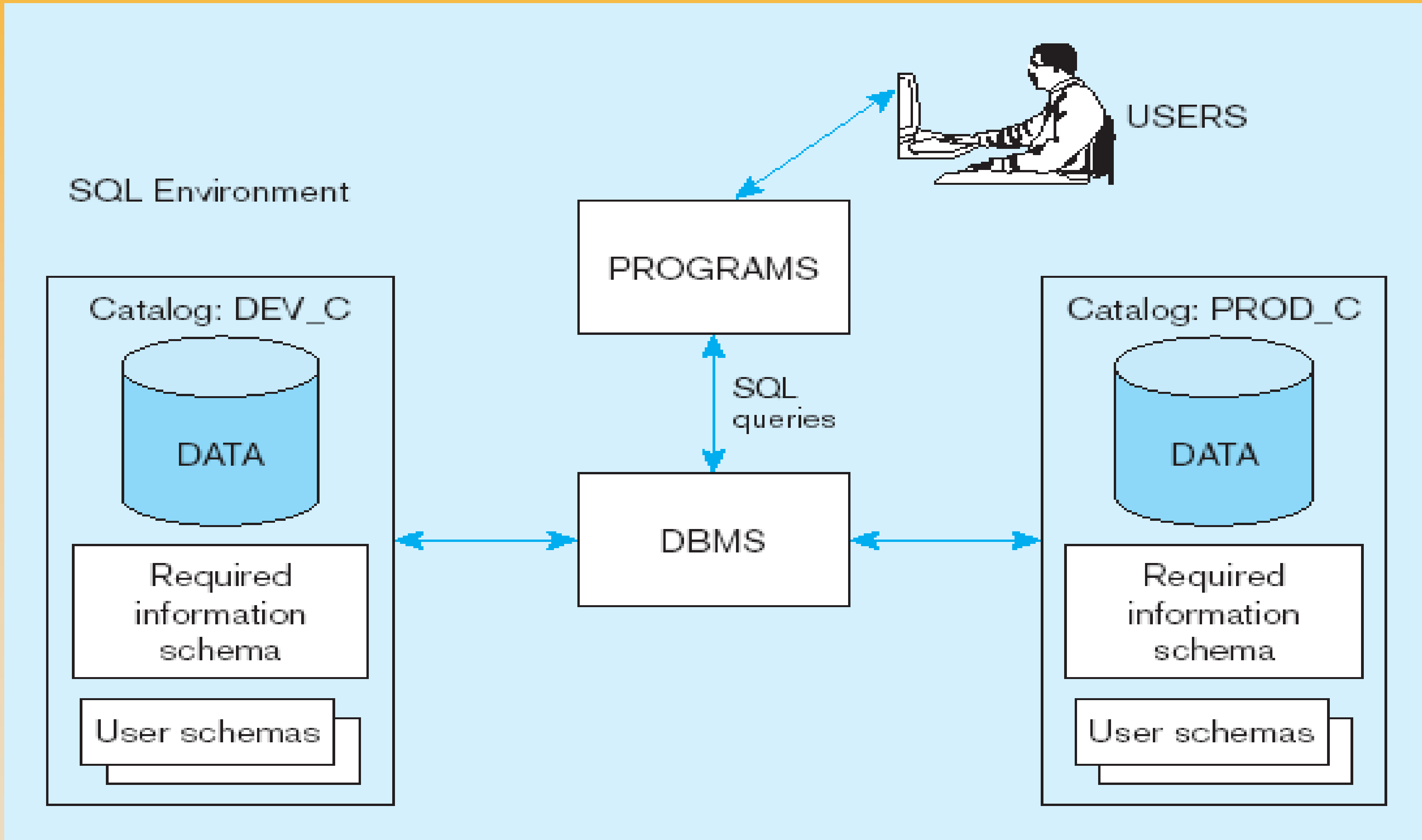
Benefits of a Standardized Relational Language

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

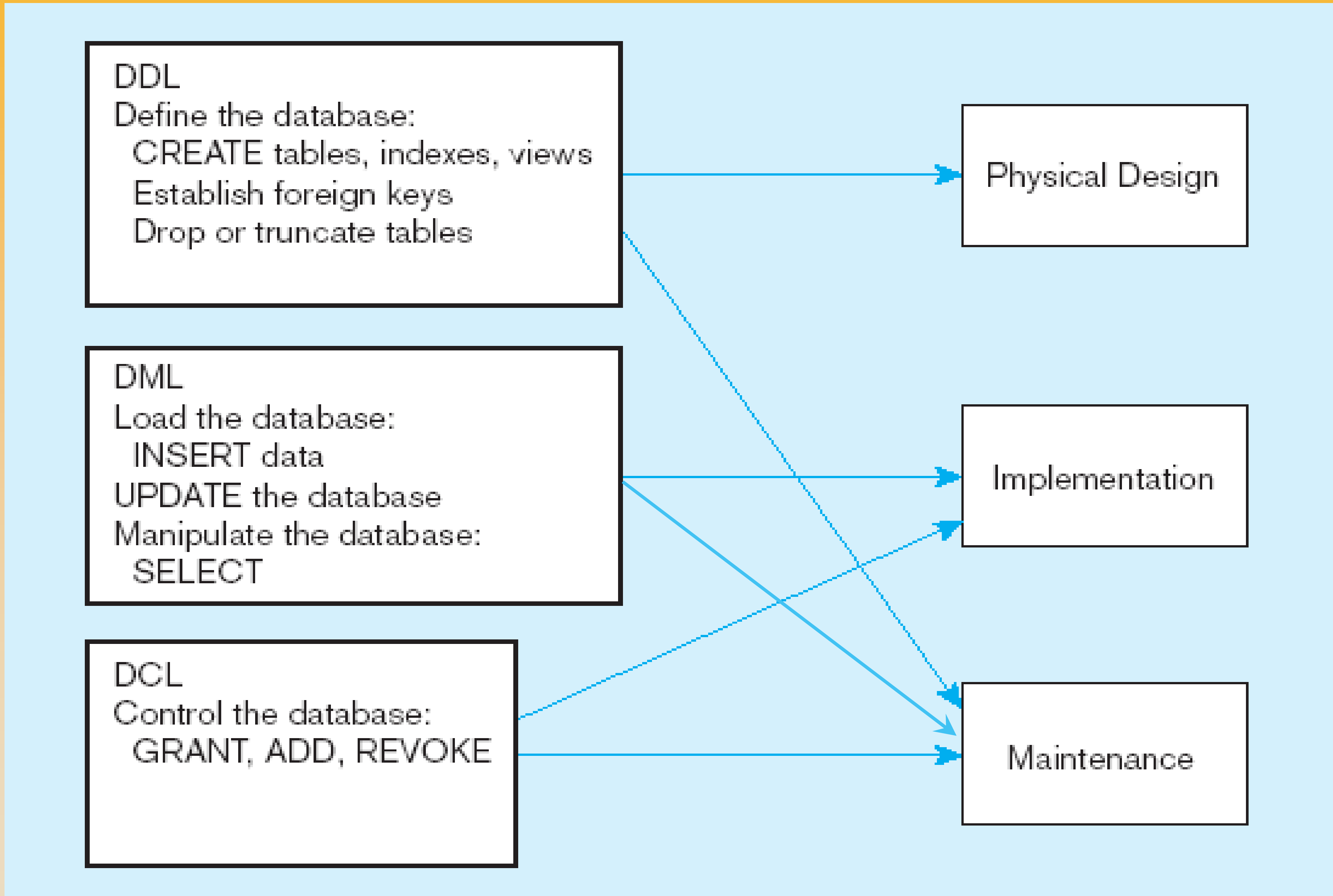
SQL Environment

- 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
- Data Manipulation Language (DML)
  - Commands that maintain and query a database
- Data Control Language (DCL)
  - Commands that control a database, including administering privileges and committing data

A simplified schematic of a typical SQL environment



DDL, DML, DCL, and the database development process



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 table and its columns
  - CREATE VIEW–defines a logical table from one or more views
- Other CREATE statements: CHARACTER SET, COLLATION, TRANSLATION, ASSERTION, DOMAIN

Table Creation

General syntax for CREATE TABLE

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*]

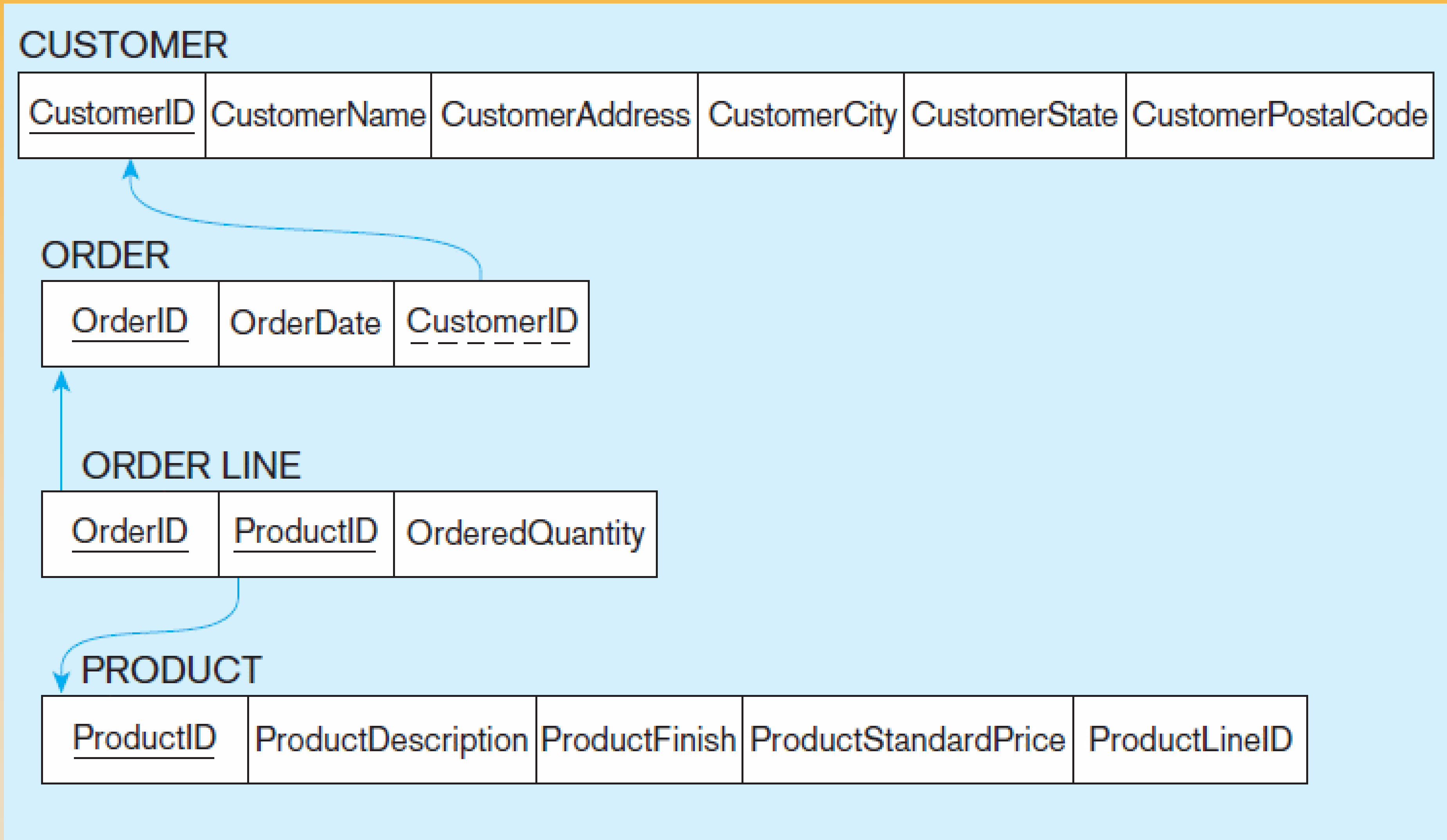
Steps in table creation:

1. Identify data types for attributes
2. Identify columns that can and cannot be null
3. Identify columns that must be unique (candidate keys)
4. Identify primary key–foreign key mates
5. Determine default values
6. Identify constraints on columns (domain specifications)
7. Create the table and associated indexes

Some 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)	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.
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	Stores a moment an event occurs, using a definable fraction of a second precision.
Boolean	BOOLEAN	Stores truth values, TRUE, FALSE, or UNKNOWN.

The following slides create tables for these table designs



SQL database definition commands for Pine Valley Furniture

```
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

```sql
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));

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));
```

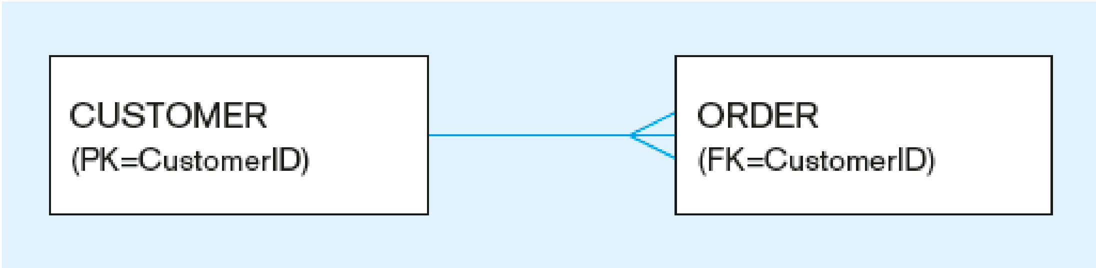
Primary key of parent table

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

Ensuring data integrity through updates



Restricted Update: A customer ID can only be deleted if it is not found in ORDER table.

```
CREATE TABLE CustomerT
        (CustomerID        INTEGER DEFAULT '999'    NOT NULL,
         CustomerName      VARCHAR(40)              NOT NULL,
         . . .
CONSTRAINT Customer_PK PRIMARY KEY (CustomerID),
ON UPDATE RESTRICT);
```

Cascaded Update: Changing a customer ID in the CUSTOMER table will result in that value changing in the ORDER table to match.

. . . ON UPDATE CASCADE);

Set Null Update: When a customer ID is changed, any customer ID in the ORDER table that matches the old customer ID is set to NULL.

. . . ON UPDATE SET NULL);

Set Default Update: When a customer ID is changed, any customer ID in the ORDER tables that matches the old customer ID is set to a predefined default value.

. . . ON UPDATE SET DEFAULT);

Relational integrity is enforced via the primary-key to foreign-key match

Changing and Removing Tables

- ALTER TABLE statement allows you to change column specifications:

  Syntax:

  - **ALTER TABLE** table_name alter_table_action;
    - **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

Command: To add a customer type column named CustomerType to the CUSTOMER table, set default value as "Commercial".

- ALTER TABLE Customer_T

  ADD COLUMN CustomerType VARCHAR(12) DEFAULT "Commercial";

Changing and Removing Tables

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

Schema Definition

- Control processing/storage efficiency:
  - Choice of indexes
  - File organizations for base tables
  - File organizations for indexes
  - Data clustering
  - Statistics maintenance
- Creating indexes
  - Speed up random/sequential access to base table data
  - Example
    - CREATE INDEX CustomerNameIdx ON Customer_T(CustomerName)
    - This makes an index for the CustomerName field of the Customer_T table

Insert Statement

- Adds data 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);
- Inserting from another table
  - INSERT INTO CA_Customer_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:200n

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 CustomerState = 'HI';
- Delete all rows
  - DELETE FROM Customer_T;

Update Statement

- Modifies data in existing rows

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

SELECT Statement

- Used for queries on single or multiple tables
- Clauses of the SELECT statement:
  - SELECT
    - List the columns (and expressions) that should 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

SELECT Example

- Find products with standard price less than $275

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

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 a Function

- Using the COUNT ***aggregate function*** to find totals

  SELECT COUNT(*) FROM OrderLine_T

  WHERE OrderID = 1004;

  Note: with aggregate functions you can't have single-valued columns included in the SELECT clause

  SELECT COUNT(*), ProductID FROM OrderLine_T

  WHERE OrderID = 1004;

  The above statement will result in error.

SELECT Example Using a Function

- Other aggregate functions: MAX, MIN, SUM, AVG

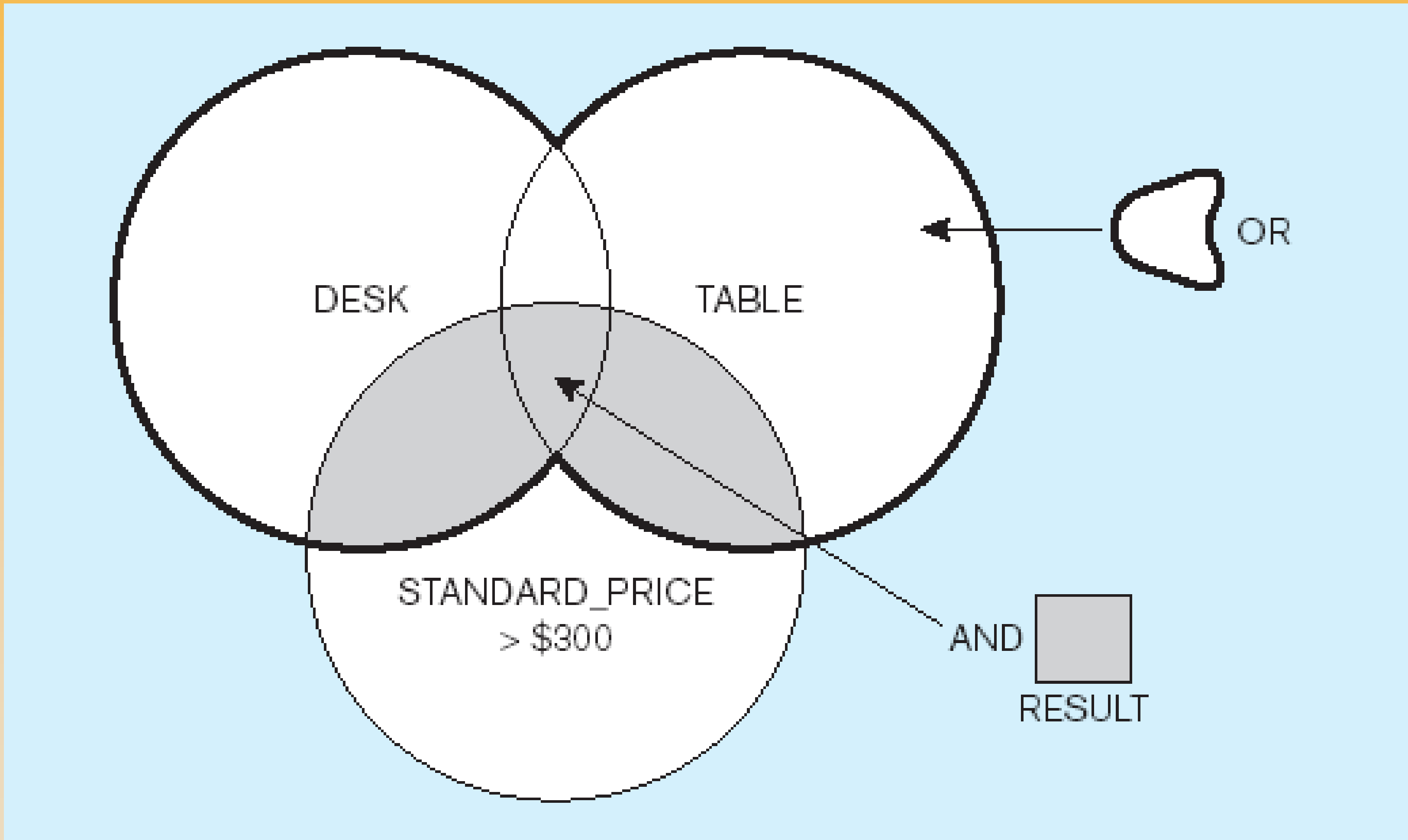
```
SELECT MAX(ProductID) FROM OrderLine_T
    WHERE OrderID = 1004;
SELECT MIN(ProductDescription)
    FROM Product_T;
```

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;
- Use * instead of % in MS-Access

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

Venn Diagram from the Previous Query



SELECT Example –
Sorting Results with the ORDER BY clause

- Sort the results first by CustomerState, and within a state by CustomerName

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

- Use DESC for sorting in descending order

  e.g., ORDER BY CustomerState DESC, 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

SELECT Example–
Categorizing Results Using the 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;
```

Note: you can use single-value fields with aggregate functions if they are included in the GROUP BY clause

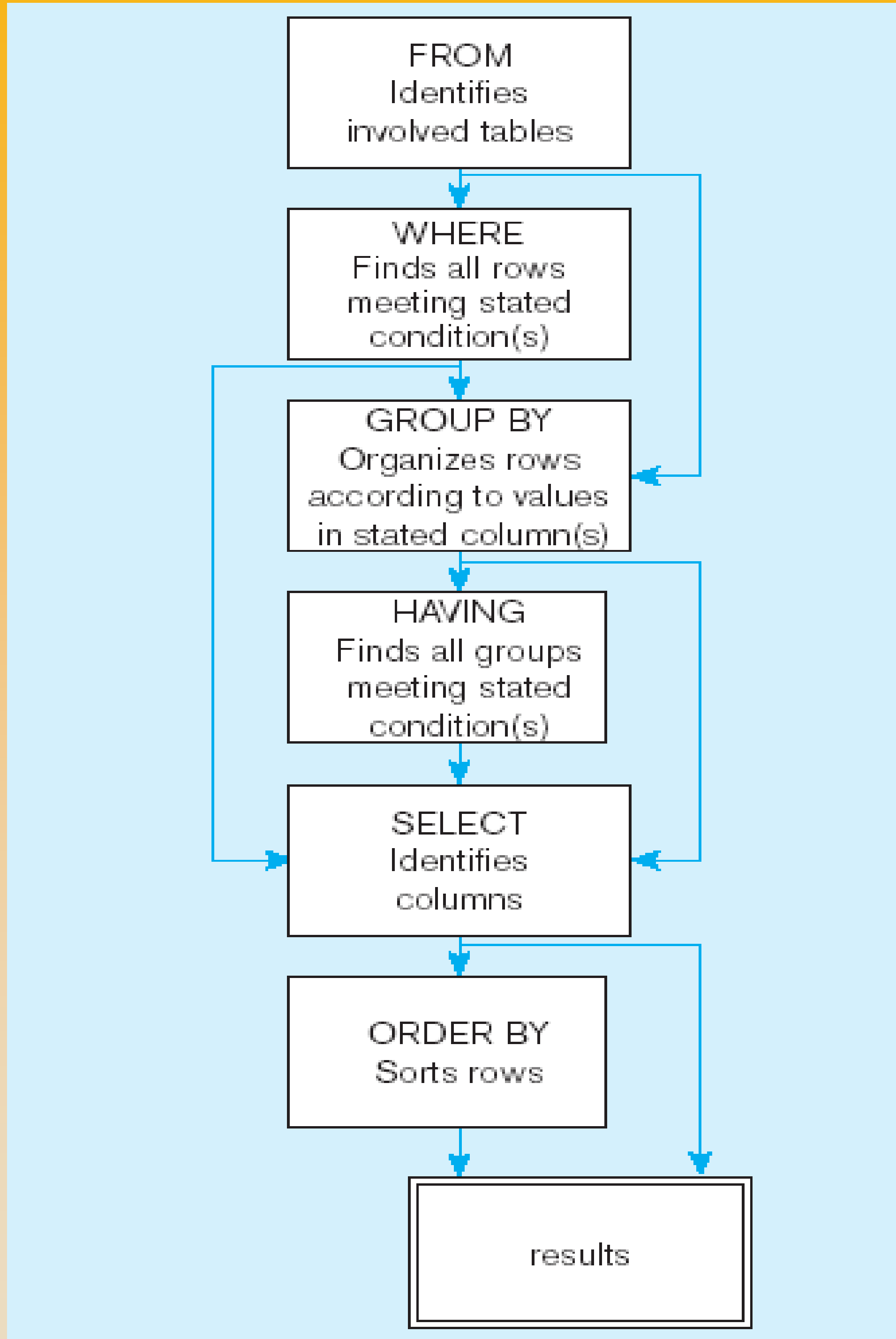
SELECT Example–
Qualifying Results by Categories
Using the HAVING Clause

- 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

SQL statement processing order (adapted from van der Lans, p.100)



Using and Defining Views

- Views provide users controlled access to tables
- Base Table – table containing the raw data

```
CREATE VIEW ExpensiveStuff_V
  AS
    SELECT ProductID, ProductDescription, ProductStandardPrice
      FROM Product_T
        WHERE ProductStandardPrice > 300
        WITH CHECK OPTION;
```

- View has a name
- View is based on a SELECT statement
- CHECK_OPTION works only for updateable views and prevents updates that would create rows not included in the view

Advantages of Views

- Simplify query commands
- Enhance programming productivity
- Assist with data security (but don't rely on views for security, there are more important security measures)
- Provide customized view for user

Dynamic vs Materialized Views

- Dynamic View
  - A “virtual table” created dynamically upon request by a user
  - No data actually stored; instead data from base table made available to user
  - Based on SQL SELECT statement on base tables or other views
- Materialized View
  - Copy or replication of data
  - Data actually stored
  - Must be refreshed periodically to match the corresponding base tables

Dynamic vs Materialized Views

- Advantages of Dynamic Views
  - Contain most current base table data
  - Use little storage space
- Disadvantages of Dynamic Views
  - Use processing time each time view is referenced
  - May or may not be directly updateable

Processing Multiple Tables–Joins

- **Join**–a relational operation that causes two or more tables with a common domain to be combined into a single table or view
- **Natural join (inner join)**–a join in which the joining condition is based on equality between values in the common columns
- **Outer join**–a join in which rows that do not have matching values in common columns are nonetheless included in the result table (as opposed to *inner* join, in which rows must have matching values in order to appear in the result table)
- **Union join**–includes all columns from each table in the join, and an instance for each row of each table

The common columns in joined tables are usually the primary key of the dominant table and the foreign key of the dependent table in 1:M relationships

Pine Valley Furniture Company Customer and Order tables with pointers from customers to their orders

Order_T

OrderID	OrderDate	CustomerID
1001	10/21/2010	1
1002	10/21/2010	8
1003	10/22/2010	15
1004	10/22/2010	5
1005	10/24/2010	3
1006	10/24/2010	2
1007	10/27/2010	11
1008	10/30/2010	12
1009	11/5/2010	4
1010	11/5/2010	1
0		0

Record: 1 of 10 No Filter

Customer_T

CustomerID	CustomerName	CustomerAddress	CustomerCity	CustomerState	CustomerPostalCode
1	Contemporary Casuals	1355 S Hines Blvd	Gainesville	FL	32601-2871
2	Value Furniture	15145 S.W. 17th St.	Plano	TX	75094-7743
3	Home Furnishings	1900 Allard Ave.	Albany	NY	12209-1125
4	Eastern Furniture	1925 Beltline Rd.	Carteret	NJ	07008-3188
5	Impressions	5585 Westcott Ct.	Sacramento	CA	94206-4056
6	Furniture Gallery	325 Flatiron Dr.	Boulder	CO	80514-4432
7	Period Furniture	394 Rainbow Dr.	Seattle	WA	97954-5589
8	Calfornia Classics	816 Peach Rd.	Santa Clara	CA	96915-7754
9	M and H Casual Furniture	3709 First Street	Clearwater	FL	34620-2314
10	Seminole Interiors	2400 Rocky Point Dr.	Seminole	FL	34646-4423
11	American Euro Lifestyles	2424 Missouri Ave N.	Prospect Park	NJ	07508-5621
12	Battle Creek Furniture	345 Capitol Ave. SW	Battle Creek	MI	49015-3401
13	Heritage Furnishings	66789 College Ave.	Carlisle	PA	17013-8834
14	Kaneohe Homes	112 Kiowai St.	Kaneohe	HI	96744-2537
15	Mountain Scenes	4132 Main Street	Ogden	UT	84403-4432
(New)					

Record: 16 of 16 No Filter Search

These tables are used in queries that follow

Natural Join Example

- For each customer who placed an order, what is the customer's name and order number?

Join involves multiple tables in FROM clause

```
SELECT Customer_T.CustomerID, CustomerName, OrderID
FROM Customer_T INNER JOIN Order_T ON
Customer_T.CustomerID = Order_T.CustomerID;
```

ON clause performs the equality check for common columns of the two tables

Note: from Fig. 1, you see that only 10 Customers have links with orders.

➔ Only 10 rows will be returned from this INNER join.

Outer Join Example

- List the customer name, ID number, and order number for all customers. Include customer information even for customers that do not have an order

```
SELECT Customer_T.CustomerID, CustomerName, OrderID
FROM Customer_T LEFT OUTER JOIN Order_T
ON Customer_T.CustomerID = Order_T.CustomerID;
```

LEFT OUTER JOIN syntax with ON causes customer data to appear even if there is no corresponding order data

Unlike INNER join, this will include customer rows with no matching order rows

Results

Unlike INNER join, this will include customer rows with no matching order rows

CUSTOMERID	CUSTOMERNAME	ORDERID
1	Contemporary Casuals	1001
1	Contemporary Casuals	1010
2	Value Furniture	1006
3	Home Furnishings	1005
4	Eastern Furniture	1009
5	Impressions	1004
6	Furniture Gallery	
7	Period Furniture	
8	California Classics	1002
9	M & H Casual Furniture	
10	Seminole Interiors	
11	American Euro Lifestyles	1007
12	Battle Creek Furniture	1008
13	Heritage Furnishings	
14	Kaneohe Homes	
15	Mountain Scenes	1003

16 rows selected.

Multiple Table Join Example

- Assemble all information necessary to create an invoice for order number 1006

Four tables involved in this join

```
SELECT Customer_T.CustomerID, CustomerName, CustomerAddress,
   CustomerCity, CustomerState, CustomerPostalCode, Order_T.OrderID,
   OrderDate, OrderedQuantity, ProductDescription, ProductStandardPrice,
   (OrderedQuantity * ProductStandardPrice)
FROM Customer_T, Order_T, OrderLine_T, Product_T

WHERE  Customer_T.CustomerID = Order_T.CustomerID
     AND Order_T.OrderID = OrderLine_T.OrderID
     AND OrderLine_T.ProductID = Product_T.ProductID
     AND Order_T.OrderID = 1006;
```

Each pair of tables requires an equality-check condition in the WHERE clause, matching primary keys against foreign keys

Self-Join Example

Query: What are the employee ID and name of each employee and the name of his or her supervisor (label the supervisor's name Manager)?

```
SELECT E.EmployeeID, E.EmployeeName, M.EmployeeName AS Manager
  FROM Employee_T E, Employee_T M
  WHERE E.EmployeeSupervisor = M.EmployeeID;
```

The same table is used on both sides of the join; distinguished using table aliases

Result:

EMPLOYEEID	EMPLOYEENAME	MANAGER
123-44-347	Jim Jason	Robert Lewis

Self-joins are usually used on tables with unary relationships

Processing Multiple Tables Using Subqueries

- Subquery–placing an inner query (SELECT statement) inside an outer query
- Options:
  - In a condition of the WHERE clause
  - As a "table" of the FROM clause
  - Within the HAVING clause
- Subqueries can be:
  - Noncorrelated–executed once for the entire outer query
  - Correlated–executed once for each row returned by the outer query

Subquery Example

- Show all customers who have placed an order

The IN operator will test to see if the CUSTOMER_ID value of a row is included in the list returned from the subquery

```
SELECT CustomerName FROM Customer_T
WHERE CustomerID IN
        (SELECT DISTINCT CustomerID FROM Order_T);
```

Subquery is embedded in parentheses. In this case it returns a list that will be used in the WHERE clause of the outer query

Result:

CUSTOMER_NAME
Contemporary Casuals
Value Furniture
Home Furnishings
Eastern Furniture
Impressions
California Classics
American Euro Lifestyles
Battle Creek Furniture
Mountain Scenes

9 rows selected.

Processing a noncorrelated subquery

```
SELECT CUSTOMER_NAME
    FROM CUSTOMER_T
        WHERE CUSTOMER_ID IN
            (SELECT DISTINCT CUSTOMER_ID
                FROM ORDER_T);
```

1. The subquery (shown in the box) is processed first and an intermediate results table created:

CUSTOMER_ID
1
8
15
5
3
2
11
12
4

9 rows selected.

No reference to data in outer query, so subquery executes once only

2. The outer query returns the requested customer information for each customer included in the intermediate results table:

CUSTOMER_NAME
Contemporary Casuals
Value Furniture
Home Furnishings
Eastern Furniture
Impressions
California Classics
American Euro Lifestyles
Battle Creek Furniture
Mountain Scenes

9 rows selected.

These are the only customers that have IDs in the ORDER_T table

1. The subquery executes and returns the customer IDs from the ORDER_T table
2. The outer query on the results of the subquery

Correlated vs. Noncorrelated Subqueries

- Noncorrelated subqueries:
  - Do not depend on data from the outer query
  - Execute once for the entire outer query
- Correlated subqueries:
  - Make use of data from the outer query
  - Execute once for each row of the outer query
  - Can use the EXISTS operator

Correlated Subquery Example

- Show all orders that include furniture finished in natural ash

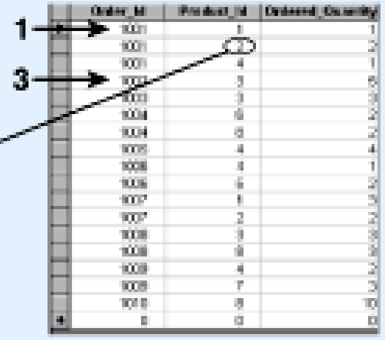
The EXISTS operator will return a TRUE value if the subquery resulted in a non-empty set, otherwise it returns a FALSE

```
SELECT DISTINCT OrderID FROM OrderLine_T
WHERE  EXISTS
   (SELECT * FROM Product_T
     WHERE ProductID = OrderLine_T.ProductID
     AND ProductFinish = 'Natural ash');
```

The subquery is testing for a value that comes from the outer query

Processing a correlated subquery

```
SELECT DISTINCT ORDER _ID FROM ORDER _LINE _T
WHERE EXISTS
        (SELECT *
          FROM PRODUCT _T
                WHERE PRODUCT _ID = ORDER _LINE _T.PRODUCT _ID
                      AND PRODUCT _FINISH = 'Natural Ash');
```



Subquery refers to outer-query data, so executes once for each row of outer query

		Product_ID	Product_Description	Product_Finish	Standard_Price	Product_Line_Id
▶	+	1	End Table	Cherry	$175.00	10001
	+	2 → 2	Coffee Table	Natural Ash	$200.00	20001
	+	4 → 3	Computer Desk	Natural Ash	$375.00	20001
	+	4	Entertainment Center	Natural Maple	$650.00	30001
	+	5	Writer's Desk	Cherry	$325.00	10001
	+	6	8-Drawer Dresser	White Ash	$750.00	20001
	+	7	Dining Table	Natural Ash	$800.00	20001
	+	8	Computer Desk	Walnut	$250.00	30001
*		(AutoNumber)			$0.00	

1. The first order ID is selected from ORDER _LINE _T: ORDER _ID =1001.
2. The subquery is evaluated to see if any product in that order has a natural ash finish. Product 2 does, and is part of the order. EXISTS is valued as *true* and the order ID is added to the result table.
3. The next order ID is selected from ORDER _LINE _T: ORDER _ID =1002.
4. The subquery is evaluated to see if the product ordered has a natural ash finish. It does. EXISTS is valued as true and the order ID is added to the result table.
5. Processing continues through each order ID. Orders 1004, 1005, and 1010 are not included in the result table because they do not include any furniture with a natural ash finish. The final result table is shown in the text on page 303.

Note: only the orders that involve products with Natural Ash will be included in the final results

Another Subquery Example

- Show all products whose standard price is higher than the average price

Subquery forms the derived table used in the FROM clause of the outer query

One column of the subquery is an aggregate function that has an alias name. That alias can then be referred to in the outer query

```
SELECT ProductDescription, ProductStandardPrice,  AvgPrice
FROM
    (SELECT AVG(ProductStandardPrice) AS AvgPrice FROM Product_T) AP,
    Product_T
    WHERE ProductStandardPrice > AvgPrice;
```

The WHERE clause normally cannot include aggregate functions, but because the aggregate is performed in the subquery its result can be used in the outer query's WHERE clause

AP is the name of the result table of the subquery.

Union Queries

- Combine the output (union of multiple queries) together into a single result table

```
SELECT C1.CUSTOMER_ID,CUSTOMER_NAME,ORDERED_QUANTITY,
'Largest Quantity' QUANTITY
   FROM CUSTOMER_T C1,ORDER_T O1, ORDER_LINE_T Q1
            WHERE C1.CUSTOMER_ID =O1.CUSTOMER_ID
            AND O1.ORDER_ID =Q1.ORDER_ID
            AND ORDERED_QUANTITY =
                    (SELECT MAX(ORDERED_QUANTITY)
                    FROM ORDER_LINE_T)
```

First query

Combine → **UNION**

```
SELECT C1.CUSTOMER_ID,CUSTOMER_NAME,ORDERED_QUANTITY,
'Smallest Quantity'
   FROM CUSTOMER_T C1,ORDER_T O1, ORDER_LINE_T Q1
           WHERE C1.CUSTOMER_ID =O1.CUSTOMER_ID
           AND O1.ORDER_ID =Q1.ORDER_ID
           AND ORDERED_QUANTITY =
                    (SELECT MIN(ORDERED_QUANTITY)
                    FROM ORDER_LINE_T)
ORDER BY ORDERED_QUANTITY;
```

Second query

Conditional Expressions Using Case Syntax

This is available with newer versions of SQL, previously not part of the standard

```
{CASE expression
{WHEN expression
THEN {expression | NULL}} . . .
 | {WHEN predicate
THEN {expression | NULL}} . . .
[ELSE {expression  NULL}]
END }
 | ( NULLIF (expression, expression) }
 | ( COALESCE (expression . . .) }
```

```
SELECT CASE
     WHEN ProductLine = 1 THEN ProductDescription
     ELSE '####'
END AS ProductDescription
FROM Product_T;
```

Tips for Developing Queries

- Be familiar with the data model (entities and relationships)
- Understand the desired results
- Know the attributes desired in result
- Identify the entities that contain desired attributes
- Review ERD
- Construct a WHERE equality for each link
- Fine tune with GROUP BY and HAVING clauses if needed
- Consider the effect on unusual data

Query Efficiency Considerations

- Instead of SELECT *, identify the specific attributes in the SELECT clause; this helps reduce network traffic of result set
- Limit the number of subqueries; try to make everything done in a single query if possible
- If data is to be used many times, make a separate query and store its results rather than performing the query repeatedly

Guidelines for Better Query Design

- Understand how indexes are used in query processing
- Keep optimizer statistics up-to-date
- Use compatible data types for fields and literals
- Write simple queries
- Break complex queries into multiple simple parts
- Don’t nest one query inside another query
- Don’t combine a query with itself (if possible avoid self-joins)
- Create temporary tables for groups of queries
- Combine update operations
- Retrieve only the data you need
- Don’t have the DBMS sort without an index
- Consider the total query processing time for ad hoc queries

Ensuring Transaction Integrity

- Transaction = A discrete unit of work that must be completely processed or not processed at all
  - May involve multiple updates
  - If any update fails, then all other updates must be cancelled
- SQL commands for transactions
  - BEGIN TRANSACTION/END TRANSACTION
    - Marks boundaries of a transaction
  - COMMIT
    - Makes all updates permanent
  - ROLLBACK
    - Cancels updates since the last COMMIT

An SQL Transaction sequence (in pseudocode)

```
BEGIN transaction

  INSERT OrderID, Orderdate, CustomerID into Order_T;

  INSERT OrderID, ProductID, OrderedQuantity into OrderLine_T;
  INSERT OrderID, ProductID, OrderedQuantity into OrderLine_T;
  INSERT OrderID, ProductID, OrderedQuantity into OrderLine_T;

END transaction
```

Valid information inserted.
COMMIT work.

All changes to data
are made permanent.

Invalid ProductID entered.

Transaction will be ABORTED.
ROLLBACK all changes made to Order_T.

All changes made to Order_T
and OrderLine_T are removed.
Database state is just as it was
before the transaction began.

Data Dictionary Facilities

- System tables that store metadata
- Users usually can view some of these tables
- Users are restricted from updating them
- Some examples in Oracle 10g/11g
  - DBA_TABLES – descriptions of tables
  - DBA_CONSTRAINTS – description of constraints
  - DBA_USERS – information about the users of the system
- Examples in Microsoft SQL Server 2008
  - sys.columns – table and column definitions
  - sys.indexes – table index information
  - sys.foreign_key_columns – details about columns in foreign key constraints

Enhancements/Extensions in Newer Standards

- User-defined data types (UDT)
  - Subclasses of standard types or an object type
- Analytical functions (for OLAP)
  - CEILING, FLOOR, SQRT, RANK, DENSE_RANK
  - WINDOW–improved numerical analysis capabilities
- New Data Types
  - BIGINT, MULTISET (collection), XML
- CREATE TABLE LIKE–create a new table similar to an existing one
- MERGE

Merge Statement

```
MERGE INTO Product_T AS PROD
USING
(SELECT ProductID, ProductDescription, ProductFinish,
ProductStandardPrice, ProductLineID FROM Purchases_T) AS PURCH
    ON (PROD.ProductID = PURCH.ProductID)
WHEN MATCHED THEN UPDATE
    PROD.ProductStandardPrice = PURCH.ProductStandardPrice
WHEN NOT MATCHED THEN INSERT
    (ProductID, ProductDescription, ProductFinish, ProductStandardPrice,
    ProductLineID)
    VALUES(PURCH.ProductID, PURCH.ProductDescription,
    PURCH.ProductFinish, PURCH.ProductStandardPrice,
      PURCH.ProductLineID);
```

Makes it easier to update a table…allows combination of Insert and Update in one statement

Useful for updating master tables with new data

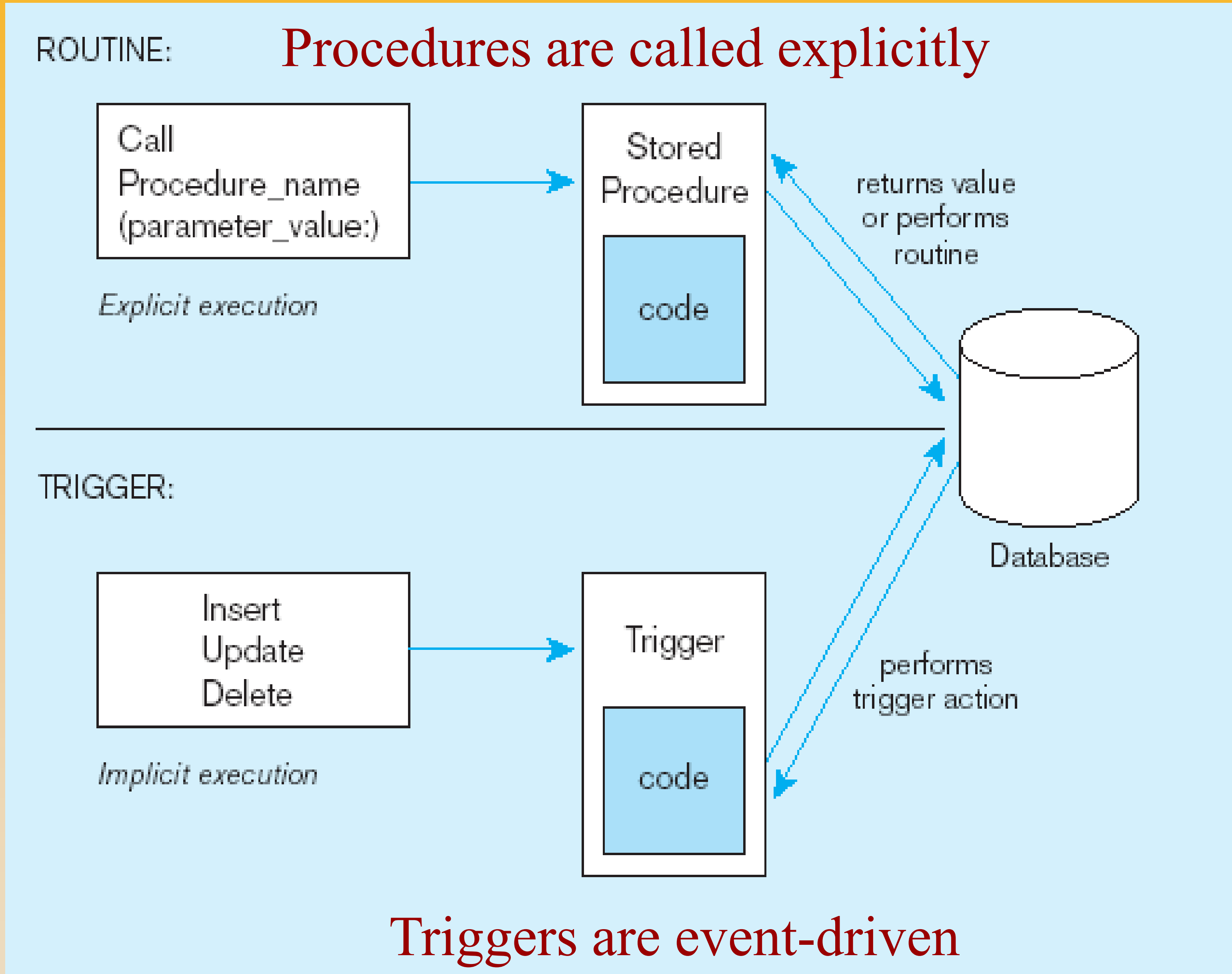
Enhancements/Extensions in Newer Standards

- Persistent Stored Modules (SQL/PSM)
  - Capability to create and drop code modules
  - New statements:
    - CASE, IF, LOOP, FOR, WHILE, etc.
    - Makes SQL into a procedural language
- Oracle has proprietary version called PL/SQL, and Microsoft SQL Server has Transact/SQL

Routines and Triggers

- **Routines**
  - Program modules that execute on demand
  - **Functions**–routines that return values and take input parameters
  - **Procedures**–routines that do not return values and can take input or output parameters
- **Triggers**
  - Routines that execute in response to a database event (INSERT, UPDATE, or DELETE)

Triggers contrasted with stored procedures



Source: adapted from Mullins, 1995.

Simplified trigger syntax

```
CREATE TRIGGER trigger_name
    {BEFORE | AFTER | INSTEAD OF} {INSERT | DELETE | UPDATE} ON
    table_name
    [FOR EACH {ROW | STATEMENT}] [WHEN (search condition)]
    <triggered SQL statement here>;
```

Create routine syntax

```
{CREATE PROCEDURE | CREATE FUNCTION} routine_name
([parameter [{,parameter} . . .]])
[RETURNS data_type result_cast]    /* for functions only */
[LANGUAGE {ADA|C|COBOL|FORTRAN|MUMPS|PASCAL|PLI|SQL}]
[PARAMETER STYLE {SQL|GENERAL}]
[SPECIFIC specific_name]
[DETERMINISTIC|NOT DETERMINISTIC]
[NO SQL|CONTAINS SQL|READS SQL DATA|MODIFIES SQL DATA]
[RETURNS NULL ON NULL INPUT|CALLED ON NULL INPUT]
[DYNAMIC RESULT SETS unsigned_integer]      /* for procedures only */
[STATIC DISPATCH]                           /* for functions only */
[NEW SAVEPOINT LEVEL | OLD SAVEPOINT LEVEL]
routine_body
```