

# CGS 2545: Database Concepts Spring 2012

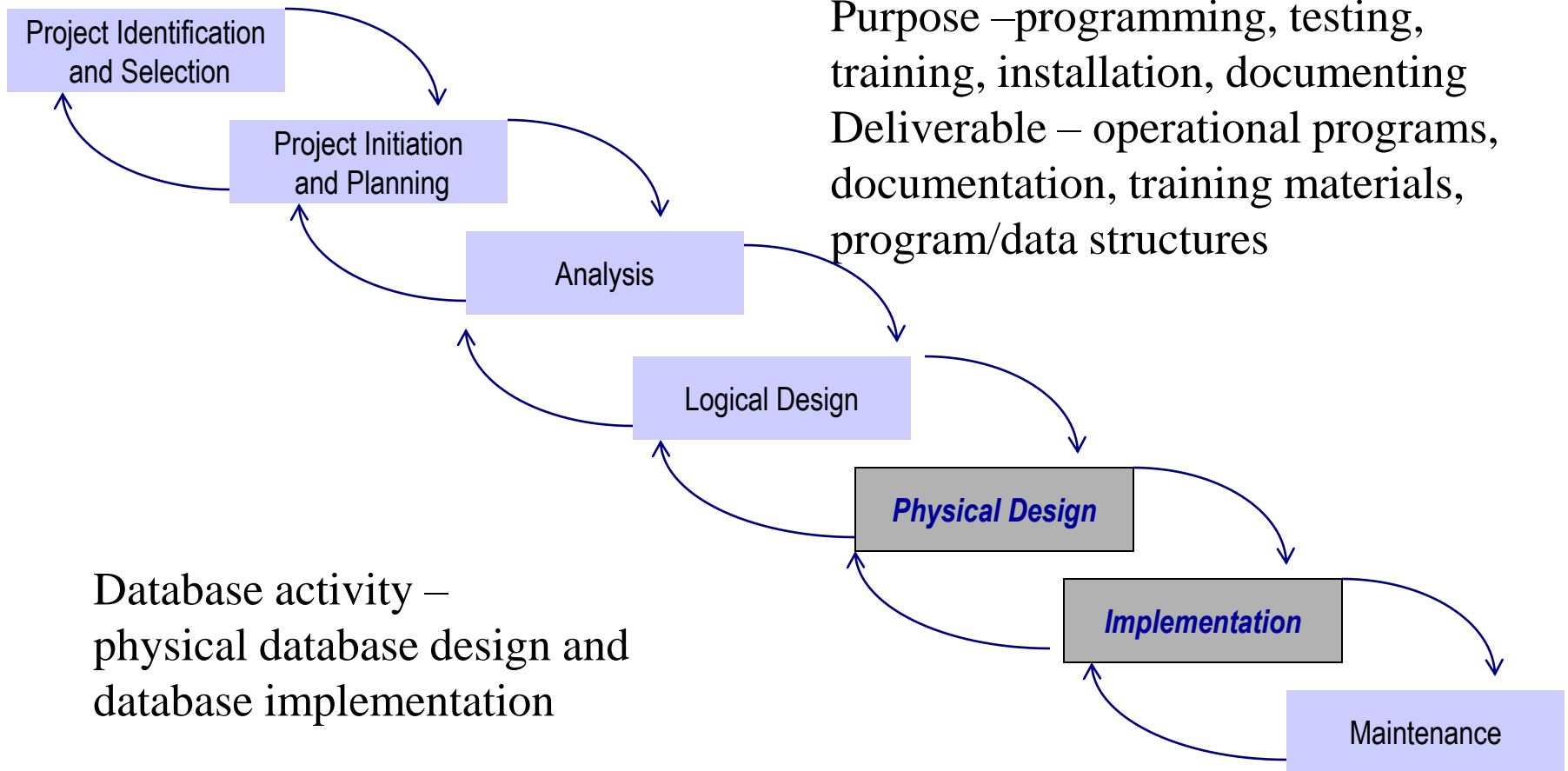
## Chapter 6 – Introduction To SQL

Instructor : Dr. Mark Llewellyn  
markl@cs.ucf.edu  
HEC 236, 407-823-2790  
<http://www.cs.ucf.edu/courses/cgs2545/spr2012>

Department of Electrical Engineering and Computer Science  
Computer Science Division  
University of Central Florida



# The Physical Design Stage of SDLC



# SQL Overview

- SQL  $\equiv$  Structured Query Language.
- The standard for relational database management systems (RDBMS).
- Current standard is SQL:2011.
  - Some previous dialects were: SQL-99, SQL:2003, and most recently SQL:2008
- Standards have a purpose:
  - Specify syntax/semantics for data definition and manipulation.
  - Define data structures.
  - Enable portability.
  - Specify minimal (core level) and complete standard.
  - 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

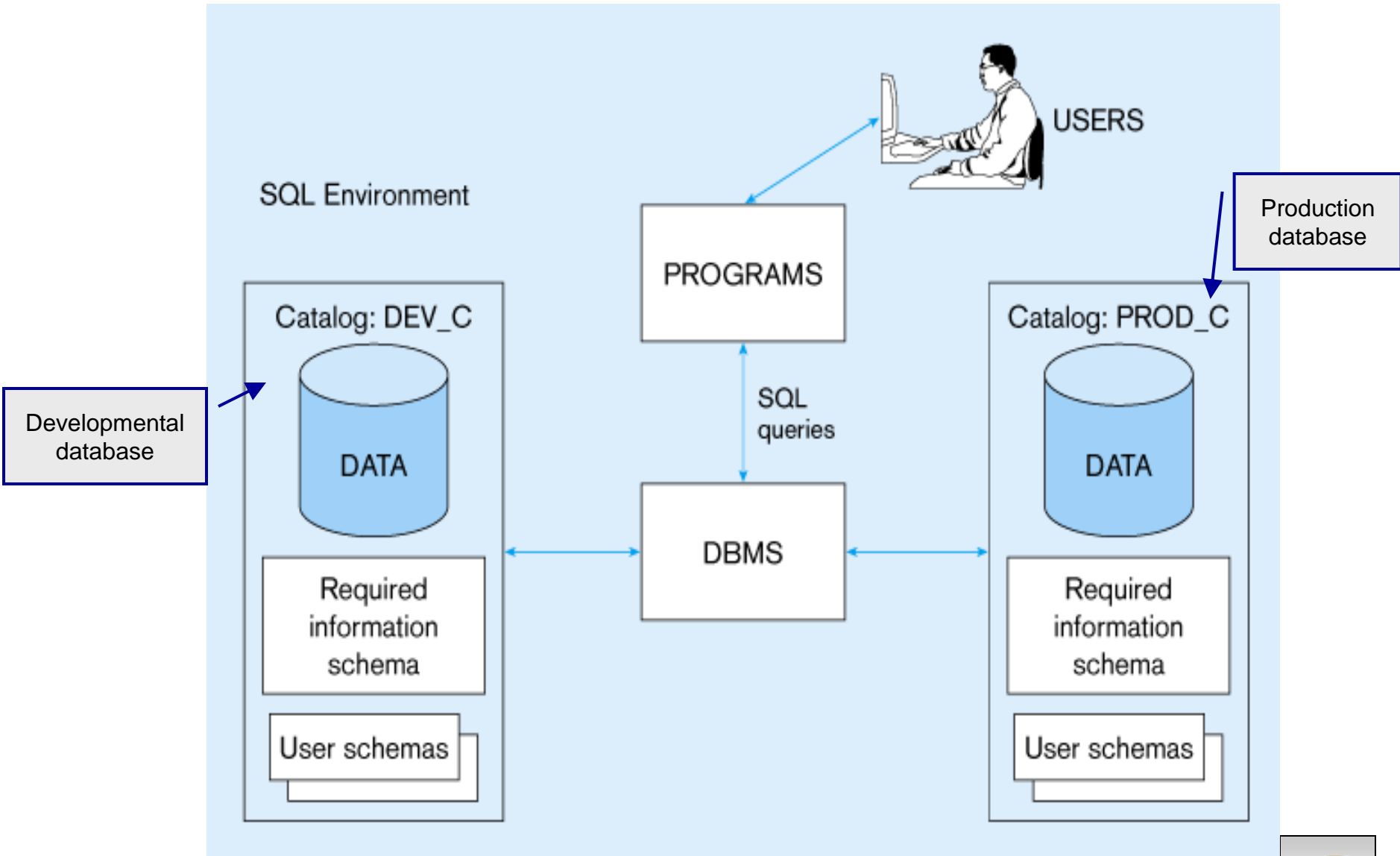


# The SQL Environment

- 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, as described by the SQL:20XX standard

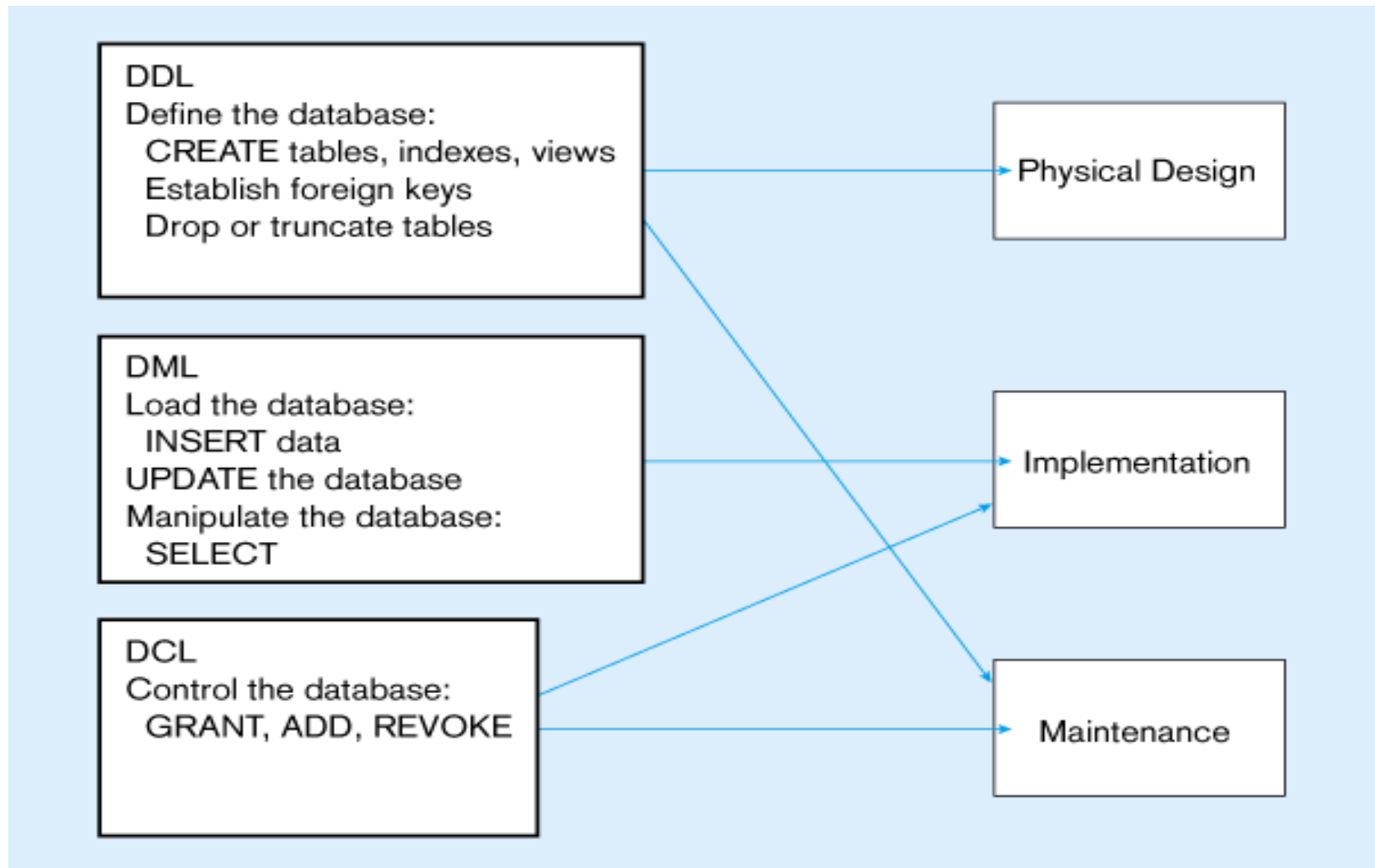


# Some SQL Data Types (from Oracle 11g)

- String types
  - CHAR(n) – fixed-length character data, n characters long  
Maximum length = 2000 bytes
  - VARCHAR2(n) – variable length character data, maximum 4000 bytes
  - LONG – variable-length character data, up to 4GB. Maximum 1 per table
- Numeric types
  - NUMBER(p,q) – general purpose numeric data type
  - INTEGER(p) – signed integer, p digits wide
  - FLOAT(p) – floating point in scientific notation with p binary digits precision
- Date/time type
  - DATE – fixed-length date/time in dd-mm-yy form



# 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 statement

```
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]
```

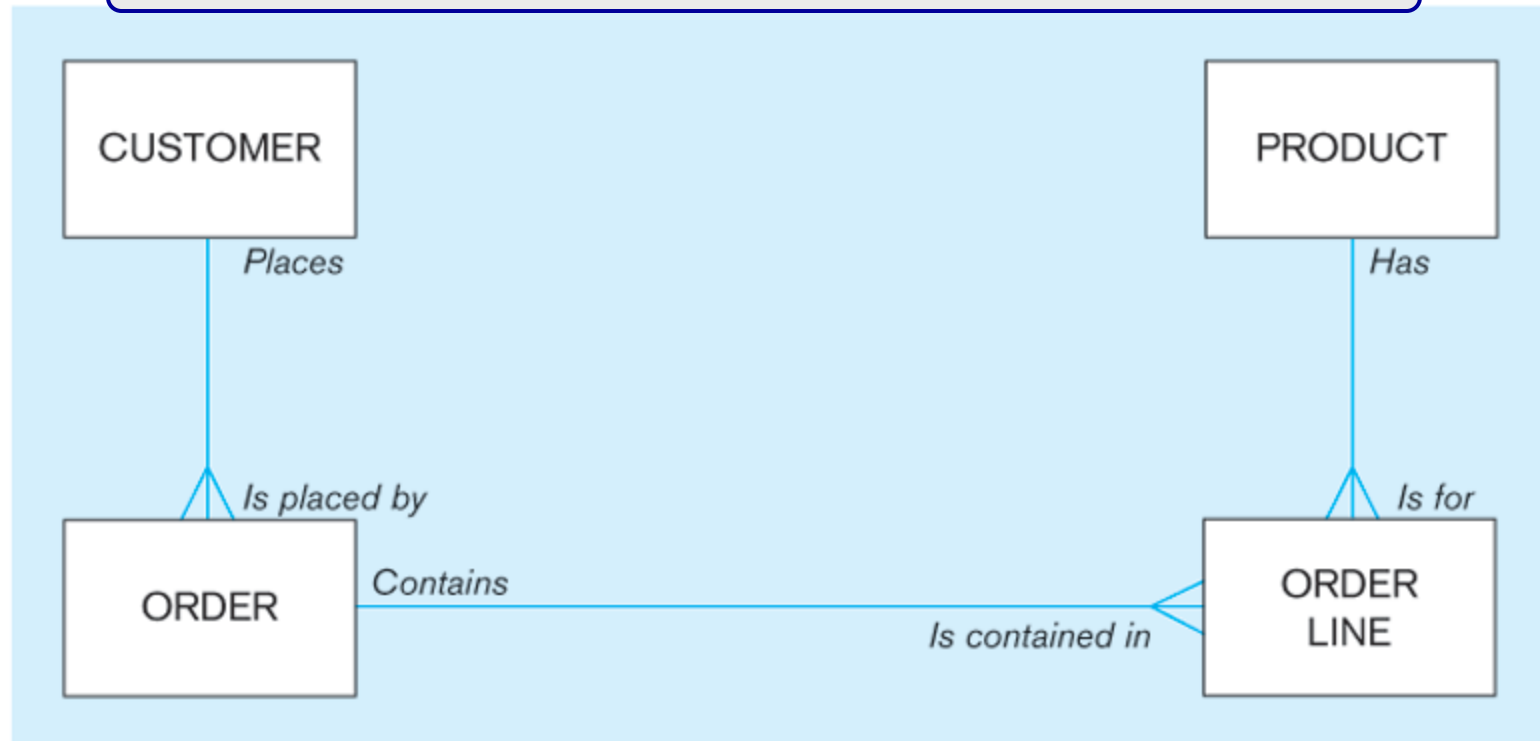
## Steps in table creation:

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



# The following few slides create tables for this enterprise data model

The Pine Valley Furniture database example from the textbook



# SQL database definition commands for Pine Valley Furniture

```
CREATE TABLE CUSTOMER_T
(CUSTOMER_ID          NUMBER(11, 0) NOT NULL,
CUSTOMER_NAME        VARCHAR2(25) NOT NULL,
CUSTOMER_ADDRESS     VARCHAR2(30),
CITY                 VARCHAR2(20),
STATE               VARCHAR2(2),
POSTAL_CODE         VARCHAR2(9),
CONSTRAINT CUSTOMER_PK PRIMARY KEY (CUSTOMER_ID));
```

Overall table  
definitions

```
CREATE TABLE ORDER_T
(ORDER_ID            NUMBER(11, 0) NOT NULL,
ORDER_DATE          DATE          DEFAULT SYSDATE,
CUSTOMER_ID        NUMBER(11, 0),
CONSTRAINT ORDER_PK PRIMARY KEY (ORDER_ID),
CONSTRAINT ORDER_FK FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER_T(CUSTOMER_ID));
```

```
CREATE TABLE PRODUCT_T
(PRODUCT_ID         INTEGER      NOT NULL,
PRODUCT_DESCRIPTION VARCHAR2(50),
PRODUCT_FINISH     VARCHAR2(20)
CHECK (PRODUCT_FINISH IN ('Cherry', 'Natural Ash', 'White Ash',
'Red Oak', 'Natural Oak', 'Walnut')),
STANDARD_PRICE     DECIMAL(6,2),
PRODUCT_LINE_ID    INTEGER,
CONSTRAINT PRODUCT_PK PRIMARY KEY (PRODUCT_ID));
```

```
CREATE TABLE ORDER_LINE_T
(ORDER_ID          NUMBER(11,0) NOT NULL,
PRODUCT_ID        NUMBER(11,0) NOT NULL,
ORDERED_QUANTITY  NUMBER(11,0),
CONSTRAINT ORDER_LINE_PK PRIMARY KEY (ORDER_ID, PRODUCT_ID),
CONSTRAINT ORDER_LINE_FK1 FOREIGN KEY(ORDER_ID) REFERENCES ORDER_T(ORDER_ID),
CONSTRAINT ORDER_LINE_FK2 FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT_T(PRODUCT_ID));
```



# Defining attributes and their data types

```
CREATE TABLE PRODUCT_T
```

```
(PRODUCT_ID          INTEGER          NOT NULL,  
 PRODUCT_DESCRIPTION VARCHAR2(50),  
 PRODUCT_FINISH      VARCHAR2(20)
```

Domain  
constraint

```
→ CHECK (PRODUCT_FINISH IN ('Cherry', 'Natural Ash', 'White Ash',  
                             'Red Oak', 'Natural Oak', 'Walnut')),
```

```
STANDARD_PRICE      DECIMAL(6,2),  
 PRODUCT_LINE_ID    INTEGER,
```

```
CONSTRAINT PRODUCT_PK PRIMARY KEY (PRODUCT_ID));
```



```
CREATE TABLE PRODUCT_T
```

```
(PRODUCT_ID
```

## Non-null specification

```
INTEGER NOT NULL,
```

```
PRODUCT_DESCRIPTION VARCHAR2(50),
```

```
PRODUCT_FINISH VARCHAR2(20)
```

```
CHECK (PRODUCT_FINISH IN ('Cherry', 'Natural Ash', 'White Ash',  
                             'Red Oak', 'Natural Oak', 'Walnut')),
```

```
STANDARD_PRICE DECIMAL(6,2),
```

```
PRODUCT_LINE_ID INTEGER,
```

```
CONSTRAINT PRODUCT_PK PRIMARY KEY (PRODUCT_ID));
```

Primary keys  
can never have  
NULL values

## Identifying primary key



```
CREATE TABLE ORDER_LINE_T
```

```
(ORDER_ID
```

```
NUMBER(11,0) NOT NULL,
```

```
PRODUCT_ID
```

```
NUMBER(11,0) NOT NULL,
```

```
ORDERED_QUANTITY
```

```
NUMBER(11,0),
```

**Non-null specifications**

```
CONSTRAINT ORDER_LINE_PK PRIMARY KEY (ORDER_ID, PRODUCT_ID),
```

**Primary key**

```
CONSTRAINT ORDER_LINE_FK1 FOREIGN KEY (ORDER_ID) REFERENCES ORDER_T (ORDER_ID),
```

```
CONSTRAINT ORDER_LINE_FK2 FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT_T (PRODUCT_ID));
```

**Some primary keys are composite –  
composed of multiple attributes**



# Controlling the values in attributes

```
CREATE TABLE ORDER_T
  (ORDER_ID          NUMBER(11,0) NOT NULL,
   ORDER_DATE       DATE          DEFAULT SYSDATE,
   CUSTOMER_ID      NUMBER(11,0),
 CONSTRAINT ORDER_PK PRIMARY KEY (ORDER_ID),
 CONSTRAINT ORDER_FK FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER_T(CUSTOMER_ID));

CREATE TABLE PRODUCT_T
  (PRODUCT_ID        INTEGER      NOT NULL,
   PRODUCT_DESCRIPTION VARCHAR2(50),
   PRODUCT_FINISH    VARCHAR2(20)
   CHECK (PRODUCT_FINISH IN ('Cherry', 'Natural Ash', 'White Ash',
                              'Red Oak', 'Natural Oak', 'Walnut')),
   STANDARD_PRICE    DECIMAL(6,2),
   PRODUCT_LINE_ID  INTEGER,
```

**Default value**

**Domain constraint**





# Identifying foreign keys and establishing relationships

```
CREATE TABLE CUSTOMER_T
```

```
(CUSTOMER_ID          NUMBER(11, 0) NOT NULL,  
  CUSTOMER_NAME       VARCHAR2(25) NOT NULL,  
  CUSTOMER_ADDRESS    VARCHAR2(30),  
  CITY                VARCHAR2(20),  
  STATE               VARCHAR2(2),  
  POSTAL_CODE         VARCHAR2(9),
```

```
CONSTRAINT CUSTOMER_PK PRIMARY KEY (CUSTOMER_ID));
```

Primary key of  
parent table

```
CREATE TABLE ORDER_T
```

```
(ORDER_ID             NUMBER(11, 0) NOT NULL,  
  ORDER_DATE          DATE          DEFAULT SYSDATE,  
  CUSTOMER_ID         NUMBER(11, 0),
```

```
CONSTRAINT ORDER_PK PRIMARY KEY (ORDER_ID),
```

```
CONSTRAINT ORDER_FK FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER_T(CUSTOMER_ID));
```

Foreign key of  
dependent table



# Some Sample Table Data For the Pine Valley Furniture Database

The screenshot shows the Microsoft Access interface with the 'Customer\_T' table open. The ribbon at the top includes 'File', 'View', 'Clipboard', 'Sort & Filter', 'Records', 'Find', 'Window', and 'Text Formatting'. The left pane shows a list of queries. The main window displays the following data:

CustomerID	CustomerName	CustomerAddress	CustomerCity	CustomerState	CustomerZip
1	Contemporary Casuals	1355 S Hines Blvd	Gainesville	FL	32601-28
2	Value Furniture	15145 S.W. 17th St.	Plano	TX	75094-77
3	Home Furnishings	1900 Allard Ave.	Albany	NY	12209-11
4	Eastern Furniture	1925 Beltline Rd.	Carteret	NJ	07008-31
5	Impressions	5585 Westcott Ct.	Sacramento	CA	94206-40
6	Furniture Gallery	325 Flatiron Dr.	Boulder	CO	80514-44
7	Period Furniture	394 Rainbow Dr.	Seattle	WA	97954-55
8	California Classics	816 Peach Rd.	Santa Clara	CA	96915-77
9	M and H Casual Furniture	3709 First Street	Clearwater	FL	34620-23
10	Seminole Interiors	2400 Rocky Point Dr.	Seminole	FL	34646-44
11	American Euro Lifestyles	2424 Missouri Ave N	Prospect Park	NJ	07508-56
12	Battle Creek Furniture	345 Capitol Ave. SW	Battle Creek	MI	49015-34
13	Heritage Furnishings	66789 College Ave.	Carlisle	PA	17013-88
14	Kaneohe Homes	112 Kiowai St.	Kaneohe	HI	96744-25
15	Mountain Scenes	4132 Main Street	Ogden	UT	84403-44
*	(New)				

Record: 5 of 15 | No Filter | Search



# Some Sample Table Data For the Pine Valley Furniture Database

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 | Search



# Some Sample Table Data For the Pine Valley Furniture Database

The screenshot displays the Microsoft Access interface. The ribbon at the top includes tabs for Views, Clipboard, Sort & Filter, Records, Find, Window, and Text Formatting. The main window shows a table named 'Product\_T' with the following data:

ProductID	ProductDescription	ProductFinis	ProductStandardPric	ProductLineID
1	End Table	Cherry	\$175.00	1
2	Coffee Table	Natural Ash	\$200.00	2
3	Computer Desk	Natural Ash	\$375.00	2
4	Entertainment Cente	Natural Maple	\$650.00	3
5	Writers Desk	Cherry	\$325.00	1
6	8-Drawer Desk	White Ash	\$750.00	2
7	Dining Table	Natural Ash	\$800.00	2
8	Computer Desk	Walnut	\$250.00	3
*	(New)		\$0.00	0

The interface also shows a left-hand pane with a list of queries and tables, including 'Product\_T: Table' and several queries like '6-p262\_Std price ... Query'. The status bar at the bottom indicates 'Record: 1 of 8' and 'No Filter'.



# Some Sample Table Data For the Pine Valley Furniture Database

The screenshot displays the Microsoft Access application window. The title bar reads "A File". The ribbon includes the following groups: Views, Clipboard, Sort & Filter (with options for Ascending, Descending, and Remove Sort), Records (with options for New, Save, Delete, and Refresh All), Find (with options for Find and Find All), Window (with options for Size to Fit Form and Switch Windows), and Text Formatting (with options for font face, size, bold, italic, underline, color, and background color). The main window shows a table named "OrderLine\_T" with the following data:

OrderID	ProductID	OrderedQuantity
1001	1	2
1001	2	2
1001	4	1
1002	3	5
1003	3	3
1004	6	2
1004	8	2
1005	4	4
1006	4	1
1006	5	2
1006	7	2
1007	1	3
1007	2	2
1008	3	3
1008	8	3
1009	4	2
1009	7	3
1010	8	10

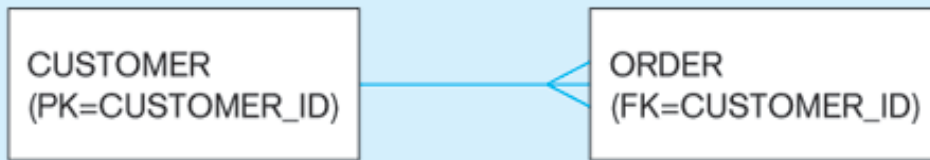
The status bar at the bottom of the table view shows "Record: 1 of 18", "No Filter", and a search box. The left-hand pane shows a list of tables and queries, including "OrderLine\_T : Table Table", "6-p265\_Count fu... Query", "6-p271\_Not DIST... Query", "6-p272\_DISTINCT... Query", "6-p273\_DISTINC... Query", "7-p296\_Invoice\_q Query", "7-p301\_Subquer... Query", "7-p302\_Subquer... Query", and "OrderData\_q Query". The bottom status bar also includes "Datasheet View" and "Num Lock".



# 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.





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

```

CREATE TABLE CUSTOMER_T
  (CUSTOMER_ID      INTEGER DEFAULT 'C999' NOT NULL,
   CUSTOMER_NAME    VARCHAR(40)      NOT NULL,
   ...
 CONSTRAINT CUSTOMER_PK PRIMARY KEY (CUSTOMER_ID),
 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);
```

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



# Restricted Update Example

## SYNTAX:

```
CREATE TABLE Customer_T ( . . .  
    CONSTRAINT Customer_PK PRIMARY KEY (customer_id), ON UPDATE RESTRICT);
```

SEMANTICS: A customer record can only be deleted from Customer\_T if they have placed no orders.

Customer\_T

customer_id	customer_name
101	Heidi
102	Frida
103	Debi
104	Claire

Deleting customer\_id = 101 would not be allowed, since that customer is related to 4 orders in Order\_T.

Deleting customer\_id = 104 would be allowed, since that customer is not related to any orders in Order\_T.

Order\_T

order_id	customer_id
10	101
12	102
14	103
16	103
18	103
20	101
22	102
24	101
26	101





# Cascaded Update Example

## SYNTAX:

```
CREATE TABLE Customer_T ( . . .  
    CONSTRAINT Customer_PK PRIMARY KEY (customer_id), ON UPDATE CASCADE);
```

SEMANTICS: Modifying a customer\_id would be reflected (cascaded) into the Order\_T table.

Assume that the initial configuration of the two relations was the same as that shown on page 24. If the Customer\_T table is modified to change customer\_id 101 to a new value of 1001, then the cascaded update would produce the updated relations as shown to the right.

Customer\_T

customer_id	customer_name
1001	Heidi
102	Frida
103	Debi
104	Claire

Order\_T

order_id	customer_id
10	1001
12	102
14	103
16	103
18	103
20	1001
22	102
24	1001
26	1001



# Set Null Update Example

## SYNTAX:

```
CREATE TABLE Customer_T ( . . .  
    CONSTRAINT Customer_PK PRIMARY KEY (customer_id), ON UPDATE SET NULL);
```

**SEMANTICS:** Modifying a customer\_id would cause any related order in the Order\_T to have the customer\_id set to null.

Assume that the initial configuration of the two relations was the same as that shown on page 24. If the Customer\_T table is modified to change customer\_id 101 to a new value of 1001, then the set null update would produce the updated relations as shown to the right.

Customer\_T

customer_id	customer_name
1001	Heidi
102	Frida
103	Debi
104	Claire

Order\_T

order_id	customer_id
10	null
12	102
14	103
16	103
18	103
20	null
22	102
24	null
26	null



# Set Default Update Example

## SYNTAX:

```
CREATE TABLE Customer_T ( . . .  
    CONSTRAINT Customer_PK PRIMARY KEY (customer_id), ON UPDATE SET DEFAULT);
```

**SEMANTICS:** Modifying a customer\_id would cause any related order in the Order\_T to have the customer\_id set to some pre-determined default value.

Assume that the initial configuration of the two relations was the same as that shown on page 24. If the Customer\_T table is modified to change customer\_id 101 to a new value of 1001, then the set default update would produce the updated relations as shown to the right. (Assume the default value was set to be 00000.)

Customer\_T

customer_id	customer_name
1001	Heidi
102	Frida
103	Debi
104	Claire

Order\_T

order_id	customer_id
10	00000
12	102
14	103
16	103
18	103
20	00000
22	102
24	00000
26	00000



# Data Integrity Controls

- Support for referential integrity constraints varies across the various RDBMSs.
- ON DELETE CASCADE is supported by Access, SQL Server, and Oracle.
- ON UPDATE CASCADE is supported by Access and SQL Server, but not Oracle.
- SET NULL is supported by Oracle, but not by Access nor SQL Server.



# Changing and Removing Tables

- **ALTER TABLE** statement allows you to change column specifications:
  - ALTER TABLE CUSTOMER\_T ADD (TYPE VARCHAR(2))
- **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 NAME\_IDX ON  
CUSTOMER\_T(CUSTOMER\_NAME)
    - This makes an index for the CUSTOMER\_NAME 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 (PRODUCT_ID, PRODUCT_DESCRIPTION, PRODUCT_FINISH, STANDARD_PRICE, PRODUCT_ON_HAND) VALUES (1, 'End Table', 'Cherry', 175, 8);`
- Inserting from another table
  - `INSERT INTO CA_CUSTOMER_T SELECT * FROM CUSTOMER_T WHERE STATE = 'CA';`



# Delete Statement

- Removes rows from a table.
- Delete certain rows
  - **DELETE FROM CUSTOMER\_T WHERE STATE = 'HI';**
- Delete all rows
  - **DELETE FROM CUSTOMER\_T;**





# Update Statement

- Modifies data in existing rows
- `UPDATE PRODUCT_T SET UNIT_PRICE = 775  
WHERE PRODUCT_ID = 7;`

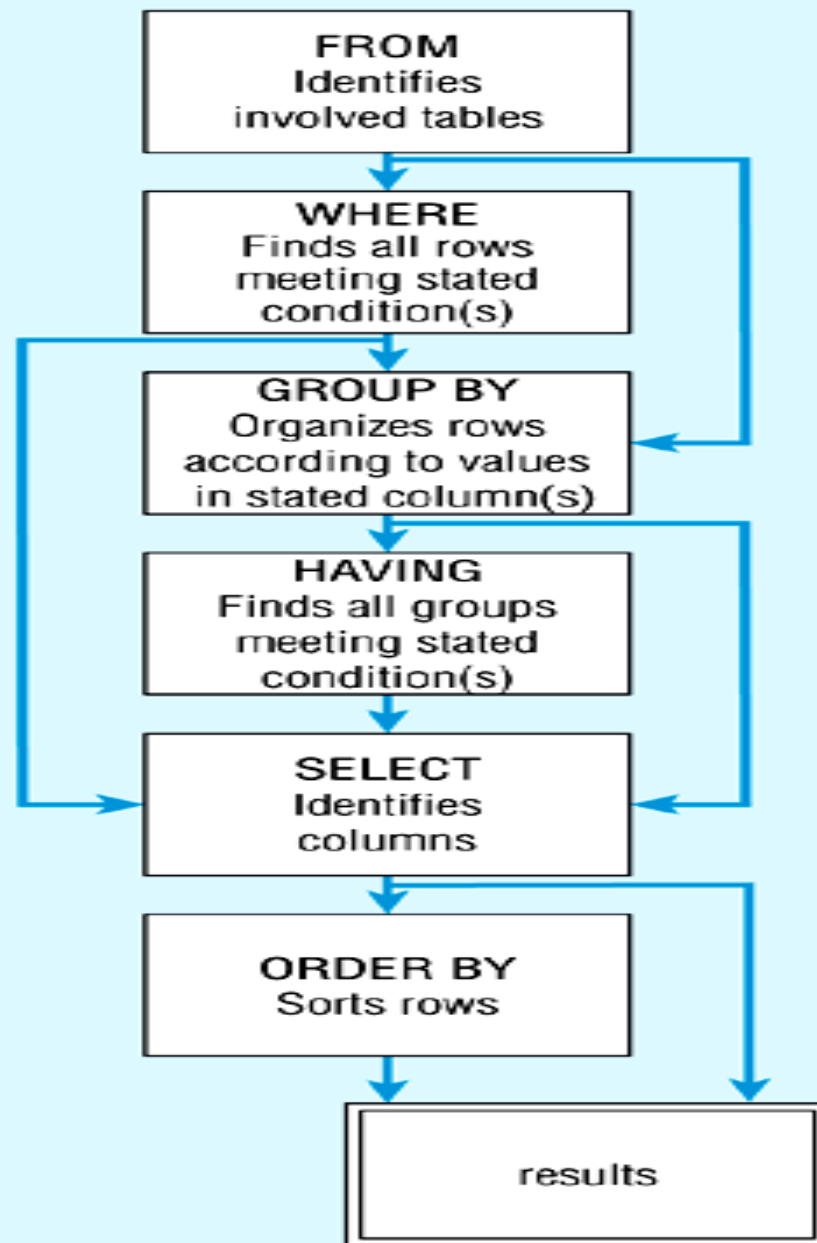


# SELECT Statement

- 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

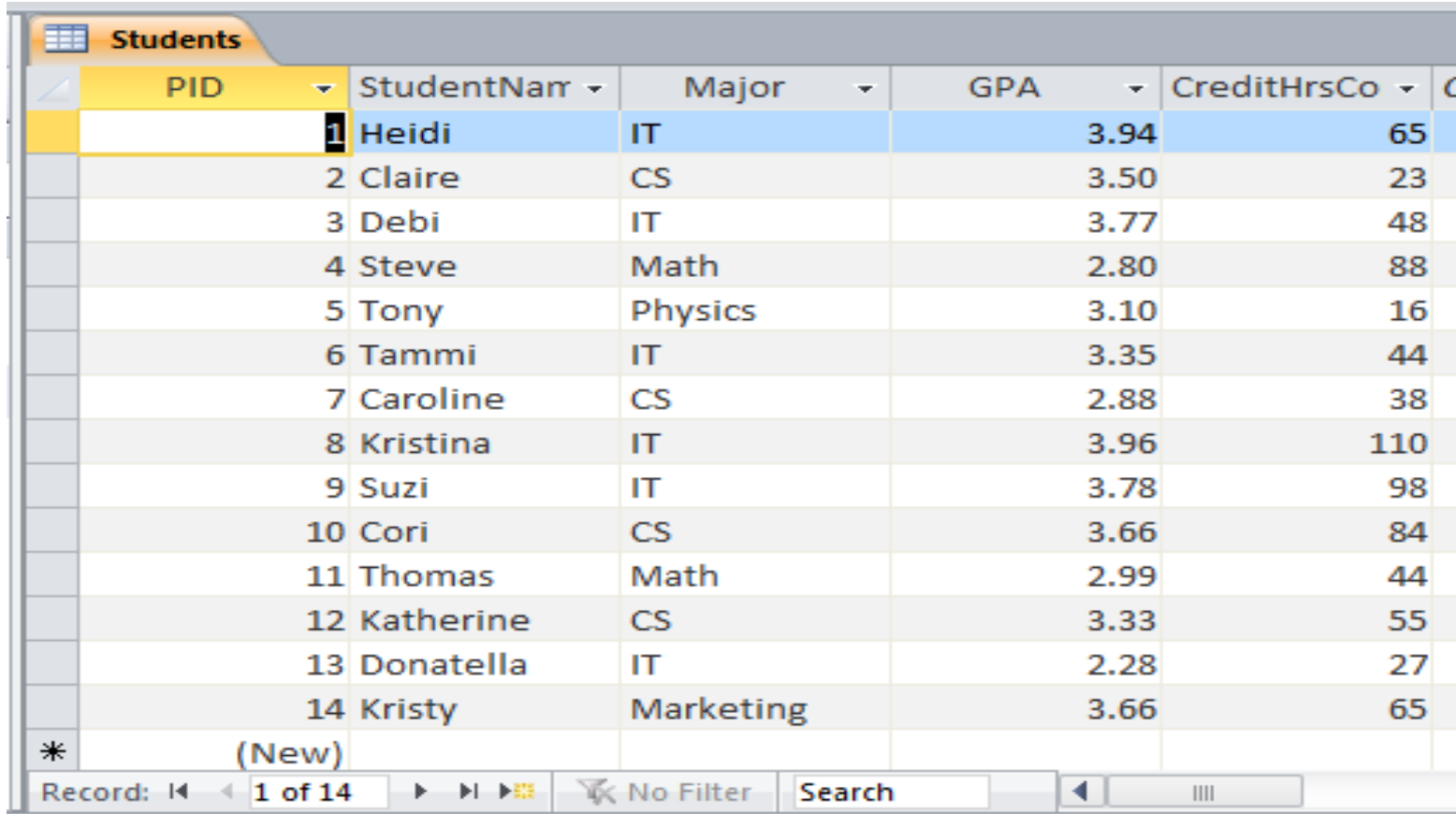


# SQL SELECT statement processing order



# Sample Database For SELECT Examples

- For the remainder of this set of notes, let's use the sample database table instance shown below:



PID	StudentName	Major	GPA	CreditHrsCo
1	Heidi	IT	3.94	65
2	Claire	CS	3.50	23
3	Debi	IT	3.77	48
4	Steve	Math	2.80	88
5	Tony	Physics	3.10	16
6	Tammi	IT	3.35	44
7	Caroline	CS	2.88	38
8	Kristina	IT	3.96	110
9	Suzi	IT	3.78	98
10	Cori	CS	3.66	84
11	Thomas	Math	2.99	44
12	Katherine	CS	3.33	55
13	Donatella	IT	2.28	27
14	Kristy	Marketing	3.66	65
*	(New)			

Record: 1 of 14 | No Filter | Search



# SELECT Example

- The most basic form of the SELECT statement is to select all attributes from a single table.
- To see all of the students in the Students table, run the following query:

```
SELECT * FROM students;
```

The \* is a wildcard character that is used in this case to represent all attributes

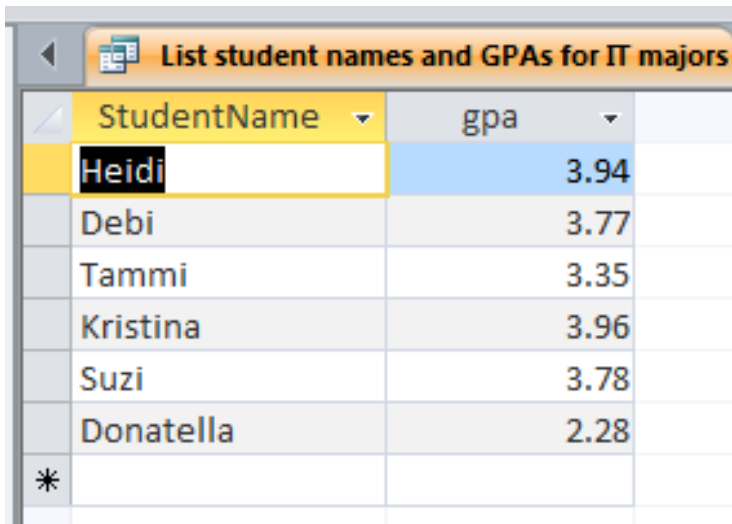
PID	StudentNar	Major	GPA	CreditHrsCo
1	Heidi	IT	3.94	65
2	Claire	CS	3.50	23
3	Debi	IT	3.77	48
4	Steve	Math	2.80	88
5	Tony	Physics	3.10	16
6	Tammi	IT	3.35	44
7	Caroline	CS	2.88	38
8	Kristina	IT	3.96	110
9	Suzi	IT	3.78	98
10	Cori	CS	3.66	84
11	Thomas	Math	2.99	44
12	Katherine	CS	3.33	55
13	Donatella	IT	2.28	27
14	Kristy	Marketing	3.66	65



# SELECT Example

- List only the names and GPAs of those students who are IT majors.

```
SELECT studentname, gpa
FROM students
WHERE major = "IT";
```



The screenshot shows a query result window titled "List student names and GPAs for IT majors". The table has two columns: "StudentName" and "gpa". The data is as follows:

StudentName	gpa
Heidi	3.94
Debi	3.77
Tammi	3.35
Kristina	3.96
Suzi	3.78
Donatella	2.28

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

All the normal comparison operators are available.



# SELECT Example using Alias

- An alias can be created as an alternative column or table name. It is useful with long names to shorten the query expression.
- Example:

Column name alias

```
Aliasing Query - lists names and credits for CS majors  
select s.studentName as sname, s.creditHrsCompleted  
from students as s  
where major="CS"
```

Table name alias

sname	creditHrsCompleted
Claire	23
Caroline	38
Cori	84
Katherine	55



# SELECT Example Using a Function

- Using the COUNT *aggregate function* to find totals

```
Count the number of IT majors
SELECT count(*)
FROM students
WHERE major = "IT";
```

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

- Aggregates can be used in the following types of clauses:
  - The WHERE clause of an ABORT statement to specify an abort condition.
  - But an aggregate function *cannot be used in the WHERE clause of a SELECT statement.*
  - A HAVING clause to specify a group condition.

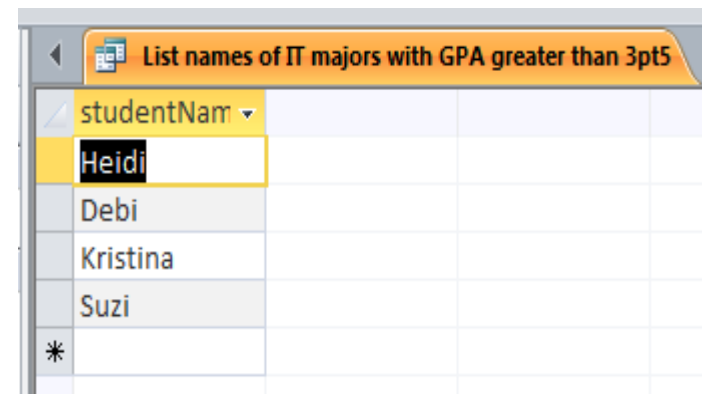




# SELECT Example – Boolean Operators

- Boolean operators **AND**, **OR**, and **NOT** are used for customizing conditions in WHERE clauses to allow for more complicated query expressions.

```
List names of IT majors with GPA greater than 3pt5  
select studentName  
from students  
where major = "IT" and gpa > 3.5
```

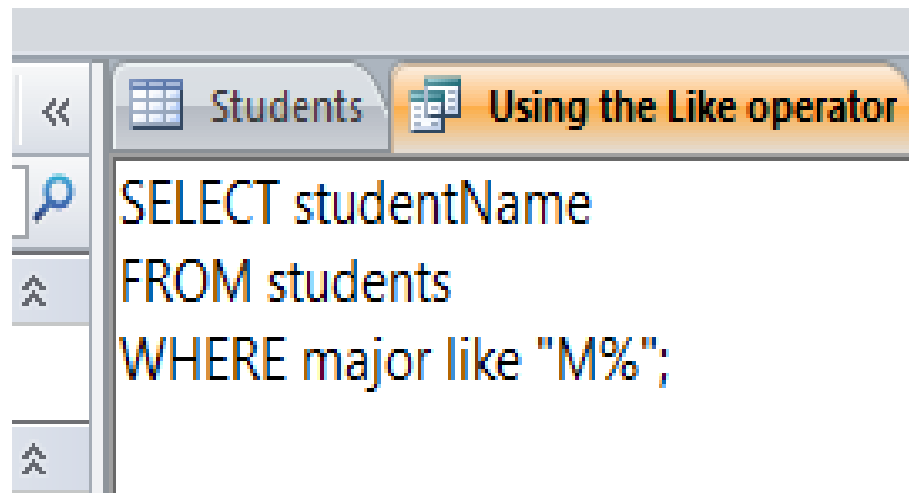


studentName
Heidi
Debi
Kristina
Suzi
*

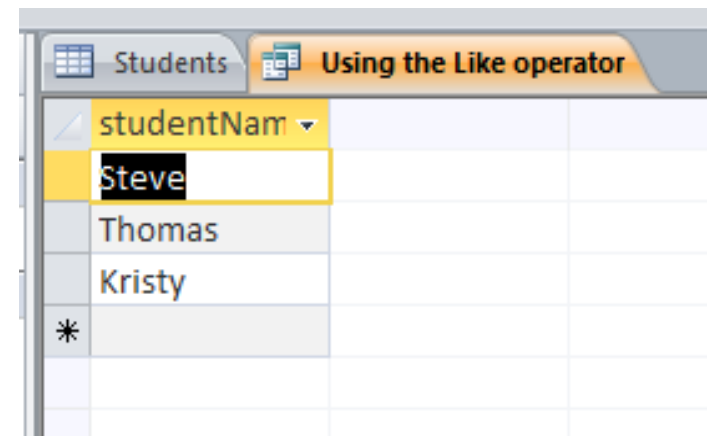


# SELECT Example – Boolean Operators

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. In the example shown below, the compare string is “M%” which will match any word that begins with the letter M.



```
SELECT studentName
FROM students
WHERE major like "M%";
```



studentName
Steve
Thomas
Kristy
*



# SELECT Example – Sorting Results with the ORDER BY Clause

- Under normal operation, most SQL implementations will return result sets in what might appear to be random order. In reality, it is based on either a key or a retrieval order. Note that in access standard result sets are ordered based on the primary key (even if the key is not part of the retrieved set of attributes).
- The ORDER BY clause is used to override the default result set ordering.
- The example on the next page illustrates this case.



```
query without order by clause
select studentName, major
from students
```

studentName	major
Heidi	IT
Claire	CS
Debi	IT
Steve	Math
Tony	Physics
Tammi	IT
Caroline	CS
Kristina	IT
Suzi	IT
Cori	CS
Thomas	Math
Katherine	CS
Donatella	IT
Kristy	Marketing

Record: 1 of 14

```
query without order by clause | query with order by clause
select studentName, major
from students
order by major
```

studentName	major
Katherine	CS
Cori	CS
Caroline	CS
Claire	CS
Donatella	IT
Suzi	IT
Kristina	IT
Tammi	IT
Debi	IT
Heidi	IT
Kristy	Marketing
Thomas	Math
Steve	Math
Tony	Physics

Record: 1 of 14



# SELECT Example – Sorting Results with the ORDER BY Clause

- In the example below the results are first ordered by major and then within the major ordered by student name.

```
using both order by and in clauses  
select studentName, major  
from students  
where major in ("CS", "IT")  
order by major, studentName
```

studentName	major
Caroline	CS
Claire	CS
Cori	CS
Katherine	CS
Debi	IT
Donatella	IT
Heidi	IT
Kristina	IT
Suzi	IT
Tammi	IT

Note: the IN operator in this example allows you to include rows whose value is included in the specified set and operationally this 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)

```
scalar aggregate query
select major, count(major)
from students
group by major
```

major	Expr1001
CS	4
IT	6
Marketing	1
Math	2
Physics	1

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



# SELECT Example –

## Qualifying Results by Category Using the HAVING Clause

```
vector aggregate query
select major, count(major)
from students
group by major
having count(major) > 1
```

major	Expr1001
CS	4
IT	6
Math	2

See next page for solution to properly name this output result field.

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



# SELECT Example –

## Qualifying Results by Category Using the HAVING Clause

```
vector aggregate query
select major, count(major) as numberOfMajors
from students
group by major
having count(major) > 1
```

major	numberOfMajors
CS	4
IT	6
Math	2

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

