The Physical Design Stage of SDLC

Purpose – programming, testing, training, installation, documenting
Deliverable – operational programs, documentation, training materials, program/data structures

Database activity – physical database design and database implementation

Project Identification and Selection

Project Initiation and Planning

Analysis

Logical Design

Physical Design

Implementation

Maintenance
SQL Overview

• SQL ≡ 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

**DDL**
- Define the database:
  - CREATE tables, indexes, views
  - Establish foreign keys
  - Drop or truncate tables

**DML**
- Load the database:
  - INSERT data
  - UPDATE the database
- Manipulate the database:
  - SELECT

**DCL**
- Control the database:
  - GRANT, ADD, REVOKE

- Physical Design
- Implementation
- Maintenance
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

Diagram:
- CUSTOMER
  - Places
- PRODUCT
  - Has
- ORDER
  - Contains
  - Is placed by
- ORDER LINE
  - Is contained in
  - Is for
### SQL database definition commands for Pine Valley Furniture

#### Overall table definitions

```sql
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));
```

```sql
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));
```

```sql
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));
```

```sql
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

```sql
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 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));

Identifying primary key

Non-null specification

Primary keys can never have NULL values
Non-null specifications

```
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));
```

Some primary keys are composite – composed of multiple attributes
Controlling the values in attributes

<table>
<thead>
<tr>
<th>CREATE TABLE ORDER_T</th>
</tr>
</thead>
<tbody>
<tr>
<td>(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));</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CREATE TABLE PRODUCT_T</th>
</tr>
</thead>
<tbody>
<tr>
<td>(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_FK FOREIGN KEY (PRODUCT_LINE_ID) REFERENCES PRODUCT_LINE_T(PRODUCT_LINE_ID));</td>
</tr>
</tbody>
</table>
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));

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));
### Some Sample Table Data For the Pine Valley Furniture Database

<table>
<thead>
<tr>
<th>CustomerID</th>
<th>CustomerName</th>
<th>CustomerAddress</th>
<th>CustomerCity</th>
<th>CustomerState</th>
<th>CustomerZip</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Contemporary Casuals</td>
<td>1355 S Hines Blvd</td>
<td>Gainesville</td>
<td>FL</td>
<td>32601-28</td>
</tr>
<tr>
<td>2</td>
<td>Value Furniture</td>
<td>15145 S.W. 17th St.</td>
<td>Plano</td>
<td>TX</td>
<td>75094-77</td>
</tr>
<tr>
<td>3</td>
<td>Home Furnishings</td>
<td>1900 Allard Ave.</td>
<td>Albany</td>
<td>NY</td>
<td>12209-11</td>
</tr>
<tr>
<td>4</td>
<td>Eastern Furniture</td>
<td>1925 Beltline Rd.</td>
<td>Carteret</td>
<td>NJ</td>
<td>07008-31</td>
</tr>
<tr>
<td>5</td>
<td>Impressions</td>
<td>5585 Westcott Ct.</td>
<td>Sacramento</td>
<td>CA</td>
<td>94206-40</td>
</tr>
<tr>
<td>6</td>
<td>Furniture Gallery</td>
<td>325 Flatiron Dr.</td>
<td>Boulder</td>
<td>CO</td>
<td>80514-44</td>
</tr>
<tr>
<td>7</td>
<td>Period Furniture</td>
<td>394 Rainbow Dr.</td>
<td>Seattle</td>
<td>WA</td>
<td>97954-55</td>
</tr>
<tr>
<td>8</td>
<td>California Classics</td>
<td>816 Peach Rd.</td>
<td>Santa Clara</td>
<td>CA</td>
<td>96915-77</td>
</tr>
<tr>
<td>9</td>
<td>M and H Casual Furniture</td>
<td>3709 First Street</td>
<td>Clearwater</td>
<td>FL</td>
<td>34620-23</td>
</tr>
<tr>
<td>10</td>
<td>Seminole Interiors</td>
<td>2400 Rocky Point Dr.</td>
<td>Seminole</td>
<td>FL</td>
<td>34664-44</td>
</tr>
<tr>
<td>11</td>
<td>American Euro Lifestyles</td>
<td>2424 Missouri Ave N</td>
<td>Prospect Park</td>
<td>NJ</td>
<td>07508-56</td>
</tr>
<tr>
<td>12</td>
<td>Battle Creek Furniture</td>
<td>345 Capitol Ave. SW</td>
<td>Battle Creek</td>
<td>MI</td>
<td>49015-34</td>
</tr>
<tr>
<td>13</td>
<td>Heritage Furnishings</td>
<td>66789 College Ave.</td>
<td>Carlisle</td>
<td>PA</td>
<td>17013-88</td>
</tr>
<tr>
<td>14</td>
<td>Kaneohe Homes</td>
<td>112 Kiowai St.</td>
<td>Kaneohe</td>
<td>HI</td>
<td>96744-25</td>
</tr>
<tr>
<td>15</td>
<td>Mountain Scenes</td>
<td>4132 Main Street</td>
<td>Ogden</td>
<td>UT</td>
<td>84403-44</td>
</tr>
</tbody>
</table>

* (New)
Some Sample Table Data For the Pine Valley Furniture Database
Some Sample Table Data For the Pine Valley Furniture Database

<table>
<thead>
<tr>
<th>ProductID</th>
<th>ProductDescription</th>
<th>Finish</th>
<th>ProductStandardPrice</th>
<th>ProductLineID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>End Table</td>
<td>Cherry</td>
<td>$175.00</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Coffee Table</td>
<td>Natural Ash</td>
<td>$200.00</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Computer Desk</td>
<td>Natural Ash</td>
<td>$375.00</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Entertainment Centre</td>
<td>Natural Maple</td>
<td>$650.00</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>Writers Desk</td>
<td>Cherry</td>
<td>$325.00</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>8-Drawer Desk</td>
<td>White Ash</td>
<td>$750.00</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>Dining Table</td>
<td>Natural Ash</td>
<td>$800.00</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>Computer Desk</td>
<td>Walnut</td>
<td>$250.00</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>(New)</td>
<td></td>
<td>$0.00</td>
<td>0</td>
</tr>
</tbody>
</table>
### Some Sample Table Data For the Pine Valley Furniture Database

<table>
<thead>
<tr>
<th>OrderID</th>
<th>ProductID</th>
<th>OrderedQuantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1001</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>1002</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>1003</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1004</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>1004</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>1005</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1006</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>1006</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>1006</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>1007</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1007</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1008</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1009</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>1009</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>1010</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</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.
Referential integrity is enforced via the primary-key to foreign-key match.

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

```sql
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.

```sql
    . . . 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.

```sql
    . . . 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.

```sql
    . . . ON UPDATE SET DEFAULT);
```
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

<table>
<thead>
<tr>
<th>customer_id</th>
<th>customer_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Heidi</td>
</tr>
<tr>
<td>102</td>
<td>Frida</td>
</tr>
<tr>
<td>103</td>
<td>Debi</td>
</tr>
<tr>
<td>104</td>
<td>Claire</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>order_id</th>
<th>customer_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>101</td>
</tr>
<tr>
<td>12</td>
<td>102</td>
</tr>
<tr>
<td>14</td>
<td>103</td>
</tr>
<tr>
<td>16</td>
<td>103</td>
</tr>
<tr>
<td>18</td>
<td>103</td>
</tr>
<tr>
<td>20</td>
<td>101</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
</tr>
<tr>
<td>24</td>
<td>101</td>
</tr>
<tr>
<td>26</td>
<td>101</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Customer_T</th>
<th>Order_T</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer_id</td>
<td>order_id</td>
</tr>
<tr>
<td>1001</td>
<td>10</td>
</tr>
<tr>
<td>102</td>
<td>12</td>
</tr>
<tr>
<td>103</td>
<td>14</td>
</tr>
<tr>
<td>104</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>1001</td>
</tr>
<tr>
<td></td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>103</td>
</tr>
<tr>
<td></td>
<td>103</td>
</tr>
<tr>
<td></td>
<td>1001</td>
</tr>
<tr>
<td></td>
<td>1001</td>
</tr>
</tbody>
</table>
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.
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**

<table>
<thead>
<tr>
<th>customer_id</th>
<th>customer_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Heidi</td>
</tr>
<tr>
<td>102</td>
<td>Frida</td>
</tr>
<tr>
<td>103</td>
<td>Debi</td>
</tr>
<tr>
<td>104</td>
<td>Claire</td>
</tr>
</tbody>
</table>

**Order_T**

<table>
<thead>
<tr>
<th>order_id</th>
<th>customer_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>00000</td>
</tr>
<tr>
<td>12</td>
<td>102</td>
</tr>
<tr>
<td>14</td>
<td>103</td>
</tr>
<tr>
<td>16</td>
<td>103</td>
</tr>
<tr>
<td>18</td>
<td>103</td>
</tr>
<tr>
<td>20</td>
<td>00000</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
</tr>
<tr>
<td>24</td>
<td>00000</td>
</tr>
<tr>
<td>26</td>
<td>00000</td>
</tr>
</tbody>
</table>
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

- \texttt{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:

<table>
<thead>
<tr>
<th>PID</th>
<th>StudentName</th>
<th>Major</th>
<th>GPA</th>
<th>CreditHrsCredits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Heidi</td>
<td>IT</td>
<td>3.94</td>
<td>65</td>
</tr>
<tr>
<td>2</td>
<td>Claire</td>
<td>CS</td>
<td>3.50</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>Debi</td>
<td>IT</td>
<td>3.77</td>
<td>48</td>
</tr>
<tr>
<td>4</td>
<td>Steve</td>
<td>Math</td>
<td>2.80</td>
<td>88</td>
</tr>
<tr>
<td>5</td>
<td>Tony</td>
<td>Physics</td>
<td>3.10</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>Tammi</td>
<td>IT</td>
<td>3.35</td>
<td>44</td>
</tr>
<tr>
<td>7</td>
<td>Caroline</td>
<td>CS</td>
<td>2.88</td>
<td>38</td>
</tr>
<tr>
<td>8</td>
<td>Kristina</td>
<td>IT</td>
<td>3.96</td>
<td>110</td>
</tr>
<tr>
<td>9</td>
<td>Suzi</td>
<td>IT</td>
<td>3.78</td>
<td>98</td>
</tr>
<tr>
<td>10</td>
<td>Cori</td>
<td>CS</td>
<td>3.66</td>
<td>84</td>
</tr>
<tr>
<td>11</td>
<td>Thomas</td>
<td>Math</td>
<td>2.99</td>
<td>44</td>
</tr>
<tr>
<td>12</td>
<td>Katherine</td>
<td>CS</td>
<td>3.33</td>
<td>55</td>
</tr>
<tr>
<td>13</td>
<td>Donatella</td>
<td>IT</td>
<td>2.28</td>
<td>27</td>
</tr>
<tr>
<td>14</td>
<td>Kristy</td>
<td>Marketing</td>
<td>3.66</td>
<td>65</td>
</tr>
</tbody>
</table>
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.
SELECT Example

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

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

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:

```
select s.studentName as sname, s.creditHrsCompleted
from students as s
where major="CS"
```
SELECT Example Using a Function

• Using the COUNT aggregate function to find totals

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 count(*)
FROM students
WHERE major = "IT";
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.

```sql
select studentName
from students
where major = "IT" and gpa > 3.5
```
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 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.
```
select studentName, major
from students
```

```
select studentName, major
from students
order by major
```
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.

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)

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

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

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.