Chapter 7
Introduction to Structured Query Language (SQL)
Objectives

In this chapter, students will learn:

• The basic commands and functions of SQL
• How to use SQL for data administration (to create tables and indexes)
• How to use SQL for data manipulation (to add, modify, delete, and retrieve data)
• How to use SQL to query a database for useful information
Introduction to SQL

• SQL functions fit into two broad categories:
  – Data definition language
  – Data manipulation language
• Basic command set has vocabulary of fewer than 100 words
• American National Standards Institute (ANSI) prescribes a standard SQL
• Several SQL dialects exist
<table>
<thead>
<tr>
<th>COMMAND OR OPTION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE SCHEMA AUTHORIZATION</td>
<td>Creates a database schema</td>
</tr>
<tr>
<td>CREATE TABLE</td>
<td>Creates a new table in the user’s database schema</td>
</tr>
<tr>
<td>NOT NULL</td>
<td>Ensures that a column will not have null values</td>
</tr>
<tr>
<td>UNIQUE</td>
<td>Ensures that a column will not have duplicate values</td>
</tr>
<tr>
<td>PRIMARY KEY</td>
<td>Defines a primary key for a table</td>
</tr>
<tr>
<td>FOREIGN KEY</td>
<td>Defines a foreign key for a table</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>Defines a default value for a column (when no value is given)</td>
</tr>
<tr>
<td>CHECK</td>
<td>Validates data in an attribute</td>
</tr>
<tr>
<td>CREATE INDEX</td>
<td>Creates an index for a table</td>
</tr>
<tr>
<td>CREATE VIEW</td>
<td>Creates a dynamic subset of rows and columns from one or more tables (see Chapter 8, Advanced SQL)</td>
</tr>
<tr>
<td>ALTER TABLE</td>
<td>Modifies a table’s definition (adds, modifies, or deletes attributes or constraints)</td>
</tr>
<tr>
<td>CREATE TABLE AS</td>
<td>Creates a new table based on a query in the user’s database schema</td>
</tr>
<tr>
<td>DROP TABLE</td>
<td>Permanently deletes a table (and its data)</td>
</tr>
<tr>
<td>DROP INDEX</td>
<td>Permanently deletes an index</td>
</tr>
<tr>
<td>DROP VIEW</td>
<td>Permanently deletes a view</td>
</tr>
<tr>
<td>COMMAND OR OPTION</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>INSERT</td>
<td>Inserts row(s) into a table</td>
</tr>
<tr>
<td>SELECT</td>
<td>Selects attributes from rows in one or more tables or views</td>
</tr>
<tr>
<td>WHERE</td>
<td>Restricts the selection of rows based on a conditional expression</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>Groups the selected rows based on one or more attributes</td>
</tr>
<tr>
<td>HAVING</td>
<td>Restricts the selection of grouped rows based on a condition</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>Orders the selected rows based on one or more attributes</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Modifies an attribute’s values in one or more table’s rows</td>
</tr>
<tr>
<td>DELETE</td>
<td>Deletes one or more rows from a table</td>
</tr>
<tr>
<td>COMMIT</td>
<td>Permanently saves data changes</td>
</tr>
<tr>
<td>ROLLBACK</td>
<td>Restores data to their original values</td>
</tr>
</tbody>
</table>

**Comparison operators**

|=, <, >, <=, >=, <>| Used in conditional expressions |

**Logical operators**

| AND/OR/NOT | Used in conditional expressions |

**Special operators**

| BETWEEN | Checks whether an attribute value is within a range |
| IS NULL | Checks whether an attribute value is null |
| LIKE | Checks whether an attribute value matches a given string pattern |
| IN | Checks whether an attribute value matches any value within a value list |
| EXISTS | Checks whether a subquery returns any rows |
| DISTINCT | Limits values to unique values |

**Aggregate functions**

| COUNT | Returns the number of rows with non-null values for a given column |
| MIN | Returns the minimum attribute value found in a given column |
| MAX | Returns the maximum attribute value found in a given column |
| SUM | Returns the sum of all values for a given column |
| AVG | Returns the average of all values for a given column |
Data Definition Commands

• The database model
  – In this chapter, a simple database with these tables is used to illustrate commands:
    • CUSTOMER
    • INVOICE
    • LINE
    • PRODUCT
    • VENDOR
  – Focus on PRODUCT and VENDOR tables
FIGURE 7.1  The database model

CUSTOMER

PK  CUS_CODE
CUS_LNAME
CUS_FNAME
CUS_INITIAL
CUS_AREAACODE
CUS_PHONE
CUS_BALANCE

INVOICE

PK  INV_NUMBER
FK1  CUS_CODE
INV_DATE

generates

LINE

PK, FK1  INV_NUMBER
PK  LINE_NUMBER
FK2  P_CODE
LINE_UNITS
LINE_PRICE

contains

is found in

VENDOR

PK  V_CODE
V_NAME
V_CONTACT
V_AREAACODE
V_PHONE
V_STATE
V_ORDER

PRODUCT

PK  P_CODE
P_DESCRIPT
P_INDATE
P_QOH
P_MIN
P_PRICE
P_DISCOUNT
V_CODE

supplies

SOURCE: Course Technology/Cengage Learning
Data Types

• Data type selection is usually dictated by nature of data and by intended use

• Supported data types:
  – Number(L,D), Integer, Smallint, Decimal(L,D)
  – Char(L), Varchar(L), Varchar2(L)
  – Date, Time, Timestamp
  – Real, Double, Float
  – Interval day to hour
  – Many other types
### Table 7.3: Data Dictionary for the Ch07_SaleCo Database

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Attribute Name</th>
<th>Contents</th>
<th>Type</th>
<th>Format</th>
<th>Range*</th>
<th>Required</th>
<th>PK or FK</th>
<th>FK Referenced Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRODUCT</td>
<td>P_CODE</td>
<td>Product code</td>
<td>VARCHAR(10)</td>
<td>XXXXXXXXXXX</td>
<td>NA</td>
<td>Y</td>
<td>PK</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P_DESCR</td>
<td>Product description</td>
<td>VARCHAR(35)</td>
<td>Xxxxxxxxxxxx</td>
<td>NA</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P_INDATE</td>
<td>Stocking date</td>
<td>DATE</td>
<td>DD-MON-YYYY</td>
<td>NA</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P_QOH</td>
<td>Units available</td>
<td>SMALLINT</td>
<td>#</td>
<td>0-9999</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P_MIN</td>
<td>Minimum units</td>
<td>SMALLINT</td>
<td>#</td>
<td>0-9999</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P_PRICE</td>
<td>Product price</td>
<td>NUMBER(8,2)</td>
<td>#.##</td>
<td>0.00-9999.00</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P_DISCOUNT</td>
<td>Discount rate</td>
<td>NUMBER(5,2)</td>
<td>0.##</td>
<td>0.00-0.20</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V_CODE</td>
<td>Vendor code</td>
<td>Vendor code</td>
<td>INTEGER</td>
<td>#</td>
<td>100-999</td>
<td>FK</td>
<td>VENDOR</td>
<td></td>
</tr>
<tr>
<td>V_NAME</td>
<td>Vendor name</td>
<td>Vendor name</td>
<td>VARCHAR(35)</td>
<td>Xxxxxxxxxxxx</td>
<td>NA</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V_CONTACT</td>
<td>Contact person</td>
<td>Contact person</td>
<td>VARCHAR(25)</td>
<td>Xxxxxxxxxxxx</td>
<td>NA</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V_AREA</td>
<td>Area code</td>
<td>Area code</td>
<td>CHAR(3)</td>
<td>999</td>
<td>NA</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V_PHONE</td>
<td>Phone number</td>
<td>Phone number</td>
<td>CHAR(8)</td>
<td>999-9999</td>
<td>NA</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V_STATE</td>
<td>State</td>
<td>State</td>
<td>CHAR(2)</td>
<td>XX</td>
<td>NA</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V_ORDER</td>
<td>Previous order</td>
<td>Previous order</td>
<td>CHAR(1)</td>
<td>X</td>
<td>Y or N</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### TABLE 7.4  Some Common SQL Data Types

<table>
<thead>
<tr>
<th>DATA TYPE</th>
<th>FORMAT</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric</td>
<td>NUMBER(L,D)</td>
<td>The declaration NUMBER(7,2) indicates that numbers will be stored with two decimal places and may be up to seven digits long, including the sign and the decimal place (for example, 12.32 or -134.99).</td>
</tr>
<tr>
<td></td>
<td>INTEGER</td>
<td>May be abbreviated as INT. Integers are (whole) counting numbers, so they cannot be used if you want to store numbers that require decimal places.</td>
</tr>
<tr>
<td></td>
<td>SMALLINT</td>
<td>Like INTEGER but limited to integer values up to six digits. If your integer values are relatively small, use SMALLINT instead of INT.</td>
</tr>
<tr>
<td></td>
<td>DECIMAL(L,D)</td>
<td>Like the NUMBER specification, but the storage length is a minimum specification. That is, greater lengths are acceptable, but smaller ones are not. DECIMAL(9,2), DECIMAL(9), and DECIMAL are all acceptable.</td>
</tr>
<tr>
<td>Character</td>
<td>CHAR(L)</td>
<td>Fixed-length character data for up to 255 characters. If you store strings that are not as long as the CHAR parameter value, the remaining spaces are left unused. Therefore, if you specify CHAR(25), strings such as Smith and Katzenjammer are each stored as 25 characters. However, a U.S. area code is always three digits long, so CHAR(3) would be appropriate if you wanted to store such codes.</td>
</tr>
<tr>
<td></td>
<td>VARCHAR(L) or VARCHAR2(L)</td>
<td>Variable-length character data. The designation VARCHAR2(25) will let you store characters up to 25 characters long. However, VARCHAR will not leave unused spaces. Oracle automatically converts VARCHAR to VARCHAR2.</td>
</tr>
<tr>
<td>Date</td>
<td>DATE</td>
<td>Stores dates in the Julian date format.</td>
</tr>
</tbody>
</table>
Creating Table Structures

• Use one line per column (attribute) definition
• Use spaces to line up attribute characteristics and constraints
• Table and attribute names are capitalized
• NOT NULL specification
• UNIQUE specification
Creating Table Structures (cont’d.)

• Primary key attributes contain both a NOT NULL and a UNIQUE specification
• RDBMS will automatically enforce referential integrity for foreign keys
• Command sequence ends with semicolon
SQL Constraints

- **NOT NULL constraint**
  - Ensures that column does not accept nulls
- **UNIQUE constraint**
  - Ensures that all values in column are unique
- **DEFAULT constraint**
  - Assigns value to attribute when a new row is added to table
- **CHECK constraint**
  - Validates data when attribute value is entered
CREATE TABLE CUSTOMER(
  CUS_CODE NUMBER PRIMARY KEY,
  CUS_LNAME VARCHAR(15) NOT NULL,
  CUS_FNAME VARCHAR(15) NOT NULL,
  CUS_INITIAL CHAR(1),
  CUS_AREACODE CHAR(3) DEFAULT '615' NOT NULL CHECK(CUS_AREACODE IN ('615', '713', '931')),
  CUS_PHONE CHAR(8) NOT NULL,
  CUS_BALANCE NUMBER(9,2) DEFAULT 0.00,
  CONSTRAINT CUS_UI1 UNIQUE(CUS_LNAME, CUS_FNAME)
);
CREATE TABLE LINE (  
  INV_NUMBER NUMBER NOT NULL,  
  LINE_NUMBER NUMBER(2,0) NOT NULL,  
  P_CODE VARCHAR(10) NOT NULL,  
  LINE_UNITS NUMBER(9,2) DEFAULT 0.00 NOT NULL,  
  LINE_PRICE NUMBER(9,2) DEFAULT 0.00 NOT NULL,  

  PRIMARY KEY (INV_NUMBER, LINE_NUMBER),  

  FOREIGN KEY (INV_NUMBER) REFERENCES INVOICE ON DELETE CASCADE,  
  FOREIGN KEY (P_CODE) REFERENCES PRODUCT(P_CODE),

  CONSTRAINT LINE_UI1 UNIQUE(INV_NUMBER, P_CODE) );
SQL Indexes

• When primary key is declared, DBMS automatically creates unique index
• Often need additional indexes
• Using CREATE INDEX command, SQL indexes can be created on basis of any selected attribute
• Composite index
  – Index based on two or more attributes
  – Often used to prevent data duplication
Data Manipulation Commands

- INSERT
- SELECT
- COMMIT
- UPDATE
- ROLLBACK
- DELETE
Adding Table Rows

• **INSERT**
  - Used to enter data into table
  - Syntax:
    - INSERT INTO columnname
      VALUES (value1, value2, … , valueN);
Adding Table Rows

- **INSERT**
  - Used to enter data into table
  - A value is required for each column
  - Syntax:
    - `INSERT INTO tablename VALUES (value1, value2, ..., valueN);`
    - `INSERT INTO CUSTOMER VALUES (10016, 'Brown', 'James', 'G', '615', '297-1228', 221.19);`
    - `INSERT INTO LINE VALUES (1003, 3, '13-Q2/P2', 5, 14.99);`

- Use **NULL** for unknown values
  - `INSERT INTO LINE VALUES (1003, 3, '13-Q2/P2', NULL, 14.99);`
Adding Table Rows (cont’d.)

• When entering values, notice that:
  – Row contents are entered between parentheses
  – Character and date values are entered between apostrophes
  – Numerical entries are not enclosed in apostrophes
  – Attribute entries are separated by commas
  – A value is required for each column

• Use NULL for unknown values
Saving Table Changes

• Changes made to table contents are not physically saved on disk until:
  – Database is closed
  – Program is closed
  – COMMIT command is used

• Syntax:
  – COMMIT [WORK];

• Will permanently save any changes made to any table in the database
Listing Table Rows

• SELECT
  – Used to list contents of table
  – Syntax:
    SELECT columnlist
    FROM tablename;

• Columnlist represents one or more attributes, separated by commas

• Asterisk can be used as wildcard character to list all attributes
Selecting Rows with Conditional Restrictions

- Select partial table contents by placing restrictions on rows to be included in output
  - Add conditional restrictions to SELECT statement, using WHERE clause

- Syntax:

  ```
  SELECT columnlist
  FROM tablelist
  [ WHERE conditionlist ] ;
  ```
- **Display all customers in the database**

```sql
SELECT * FROM CUSTOMER
```

- **Display codes and balances for all customers in the database**
• **Display codes and balances for all customers in the database**

```sql
SELECT CUS_CODE, CUS_BALANCE
FROM CUSTOMER
```

• **Display codes and balances for all customers residing in area code ‘612’ whose balances exceed $1000**
• Display codes and balances for all customers residing in area code ‘612’ whose balances exceed $1000

```sql
SELECT CUS_CODE, CUS_BALANCE
FROM CUSTOMER
WHERE CUS_AREAACODE = '612' AND CUS_BALANCE > 1000
```
Display codes and balances for all customers whose last names start with ‘Pa’ : %

```sql
SELECT CUS_CODE, CUS_BALANCE
FROM CUSTOMER
WHERE CUS_LNAME LIKE 'Pa%'
```
• For each invoice line, display the invoice number, line number and total cost

SELECT INV_NUMBER, LINE_NUMBER, LINE_PRICE*LINE_UNITS as TOTAL_COST
FROM LINE
Special Operators

- **BETWEEN**: checks whether attribute value is within a range
- **IS NULL**: checks whether attribute value is null
- **LIKE**: checks whether attribute value matches given string pattern
- **IN**: checks whether attribute value matches any value within a value list
- **EXISTS**: checks if subquery returns any rows
Find the codes for customers who don’t have a middle initial residing in area codes ‘612’, ‘617’, ‘921’ or ‘321’

```
SELECT CUS_CODE
FROM CUSTOMER
WHERE CUS_INITIAL IS NULL AND (CUS_AREAACODE='612' OR CUS_AREAACODE='617' OR CUS_AREAACODE='921' OR CUS_AREAACODE='321')
```
Find the codes for customers who don’t have a middle initial residing in area codes ‘612’, ‘617’, ‘921’ or ‘321’

SELECT CUS_CODE
FROM CUSTOMER
WHERE CUS_INITIAL IS NULL AND CUS_AREAACODE IN ('612', '617', '921', '321')
• For each invoice line, display the invoice number, line number and total cost

SELECT INV_NUMBER, LINE_NUMBER, LINE_PRICE*LINE_UNITS as TOTAL_COST
FROM LINE

• For each invoice line, display the invoice date, invoice number, line number and total cost
For each invoice line, display the invoice date, invoice number, line number and total cost

```
SELECT INV_DATE, LINE.INV_NUMBER, LINE_NUMBER, LINE_PRICE*LINE_UNITS as TOTAL_COST
FROM LINE, INVOICE
WHERE LINE.INV_NUMBER = INVOICE.INV_NUMBER
```
Find all invoice numbers for customers who reside in area codes ‘612’, ‘617’, ‘921’ or ‘321’ --- use a JOIN

SELECT INV_NUMBER
FROM INVOICE, CUSTOMER
WHERE INVOICE.CUS_CODE=CUSTOMER.CUS_CODE AND CUS_AREACODE IN (‘612’, ‘617’, ‘921’, ‘321’)
Find all invoice numbers for customers who reside in area codes ‘612’, ‘617’, ‘921’ or ‘321’ --- use an inner query with an IN

```
SELECT INV_NUMBER
FROM INVOICE
WHERE CUS_CODE IN (
  SELECT CUS_CODE
  FROM CUSTOMER
  WHERE CUS_AREACODE IN ('612', '617', '921', '321'))
```

use an inner query with EXISTS
Find all invoice numbers for customers who reside in area codes ‘612’, ‘617’, ‘921’ or ‘321’

```
SELECT INV_NUMBER
FROM INVOICE
WHERE EXISTS(
    SELECT * 
    FROM CUSTOMER
    WHERE INVOICE.CUS_CODE=CUSTOMER.CUS_CODE AND CUS_AREAACODE IN ('612','617','921','321'))
```

Find all invoice numbers for customers who DO NOT reside in area codes ‘612’, ‘617’, ‘921’ or ‘321’
Find all invoice numbers for customers who DO NOT reside in area codes ‘612’, ‘617’, ‘921’ or ‘321’

```
SELECT INV_NUMBER
FROM INVOICE
WHERE NOT EXISTS(
    SELECT * 
    FROM CUSTOMER
    WHERE INVOICE.CUS_CODE=CUSTOMER.CUS_CODE 
    AND CUS_AREACODE IN ('612', '617', '921', '321')
)
```
For each invoice line, display the invoice date, invoice number, line number and total cost. Order output by invoice date

```
SELECT  INV_DATE, LINE.INV_NUMBER, LINE_NUMBER,
        LINE_PRICE*LINE_UNITS as TOTAL_COST
FROM     LINE, INVOICE
WHERE    LINE.INV_NUMBER = INVOICE.INV_NUMBER
ORDER BY INV_DATE
```

Descending order? Within date, order by invoice number?
• **List codes for all ordered products. Do not include duplicates.**

```sql
SELECT DISTINCT (P_CODE) 
FROM LINE
```

*Oracle places NULLS it at the bottom*
Summary So Far

• Syntax:

```
SELECT columnlist
FROM tablelist
[WHERE conditionlist]
[ORDER BY columnlist [ASC | DESC]];
```

• Ascending order by default
Updating Table Rows

- **UPDATE**
  - Modify data in a table
  - Syntax:
    ```
    UPDATE tablename
    SET columnname = expression [, columnname = expression]
    [WHERE conditionlist];
    ```
- If more than one attribute is to be updated in row, separate corrections with commas
Restoring Table Contents

• ROLLBACK
  – Undoes changes since last COMMIT
  – Brings data back to prechange values
• Syntax:
  ROLLBACK;
• COMMIT and ROLLBACK only work with commands to add, modify, or delete table rows
Deleting Table Rows

• **DELETE**
  – Deletes a table row
  – Syntax:
    ```
    DELETE FROM tablename
    [WHERE conditionlist ];
    ```
• **WHERE** condition is optional
• If **WHERE** condition is not specified, all rows from specified table will be deleted
Inserting Table Rows with a SELECT Subquery

- **INSERT**
  - Inserts multiple rows from another table (source)
  - Uses SELECT subquery
  - Subquery: query embedded (or nested or inner) inside another query
  - Subquery executed first
  - Syntax:
    
    ```
    INSERT INTO tablename SELECT columnlist
    FROM tablename;
    ```
SELECT Queries

• Fine-tune SELECT command by adding restrictions to search criteria using:
  – Conditional restrictions
  – Arithmetic operators
  – Logical operators
  – Special operators
Selecting Rows with Conditional Restrictions

- Select partial table contents by placing restrictions on rows to be included in output
  - Add conditional restrictions to SELECT statement, using WHERE clause

- Syntax:

  ```sql
  SELECT columnlist
  FROM tablelist
  [ WHERE conditionlist ] ;
  ```
FIGURE 7.5 The Microsoft Access QBE and its SQL

Microsoft Access-generated SQL

User-entered SQL

SOURCE: Course Technology/Cengage Learning
<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;&gt; or !=</td>
<td>Not equal to</td>
</tr>
</tbody>
</table>
Selecting Rows with Conditional Restrictions (cont’d.)

• Using comparison operators on dates
  – Date procedures are often more software-specific than other SQL procedures

• Using computed columns and column aliases
  – SQL accepts any valid expressions (or formulas) in the computed columns
  – Alias
    • Alternate name given to a column or table in any SQL statement
Arithmetic Operators: The Rule of Precedence

- Perform operations within parentheses
- Perform power operations
- Perform multiplications and divisions
- Perform additions and subtractions

**TABLE 7.7**
The Arithmetic Operators

<table>
<thead>
<tr>
<th>ARITHMETIC OPERATOR</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Add</td>
</tr>
<tr>
<td>-</td>
<td>Subtract</td>
</tr>
<tr>
<td>*</td>
<td>Multiply</td>
</tr>
<tr>
<td>/</td>
<td>Divide</td>
</tr>
<tr>
<td>^</td>
<td>Raise to the power of (some applications use ** instead of ^)</td>
</tr>
</tbody>
</table>
Logical Operators: AND, OR, and NOT

• Searching data involves multiple conditions
• Logical operators: AND, OR, and NOT
• Can be combined
  – Parentheses enforce precedence order
    • Conditions in parentheses are always executed first
• Boolean algebra: mathematical field dedicated to use of logical operators
• NOT negates result of conditional expression
Advanced Data Definition Commands

• All changes in table structure are made by using ALTER command
• Three options:
  – ADD adds a column
  – MODIFY changes column characteristics
  – DROP deletes a column
• Can also be used to:
  – Add table constraints
  – Remove table constraints
Changing a Column’s Data Type

• ALTER can be used to change data type
• Some RDBMSs do not permit changes to data types unless column is empty
Changing a Column’s Data Characteristics

• Use ALTER to change data characteristics
• Changes in column’s characteristics are permitted if changes do not alter the existing data type
Adding a Column
Dropping a Column

• Use ALTER to add column
  – Do not include the NOT NULL clause for new column

• Use ALTER to drop column
  – Some RDBMSs impose restrictions on the deletion of an attribute
Advanced Data Updates

- **UPDATE** command updates only data in existing rows
- If relationship between entries and existing columns, can assign values to slots
- Arithmetic operators are useful in data updates
- In Oracle, **ROLLBACK** command undoes changes made by last two **UPDATE** statements
Copying Parts of Tables

- SQL permits copying contents of selected table columns
  - Data need not be reentered manually into newly created table(s)
- First create the table structure
- Next add rows to new table using table rows from another table
Adding Primary and Foreign Key Designations

• When table is copied, integrity rules do not copy
  – Primary and foreign keys are manually defined on new table

• User ALTER TABLE command
  – Syntax:
    • ALTER TABLE tablename ADD PRIMARY KEY(fieldname);
  – For foreign key, use FOREIGN KEY in place of PRIMARY KEY
Additional SELECT Query Keywords

• Logical operators work well in the query environment
• SQL provides useful functions that:
  – Count
  – Find minimum and maximum values
  – Calculate averages, etc.
• SQL allows user to limit queries to:
  – Entries having no duplicates
  – Entries whose duplicates may be grouped
Ordering a Listing

• ORDER BY clause is useful when listing order is important
• Syntax:

  SELECT columnlist
  FROM tablelist
  [WHERE conditionlist]
  [ORDER BY columnlist [ASC | DESC]];
• Ascending order by default
Listing Unique Values

- DISTINCT clause produces list of only values that are different from one another
- Example:
  ```sql
  SELECT DISTINCT V_CODE
  FROM PRODUCT;
  ```
- Access places nulls at the top of the list
  - Oracle places it at the bottom
  - Placement of nulls does not affect list contents
Aggregate Functions

- **COUNT** function tallies number of non-null values of an attribute
  - Takes one parameter: usually a column name
- **MAX** and **MIN** find highest (lowest) value in a table
  - Compute MAX value in inner query
  - Compare to each value returned by the query
- **SUM** computes total sum for any specified attribute
- **AVG** function format is similar to **MIN** and **MAX**
• **Find the largest units ordered on any line**

```sql
SELECT MAX(LINE_UNITS)
FROM LINE
```

• **Find the total number of products ordered**

```sql
SELECT COUNT(P_CODE)
FROM LINE
```

• **Find the total number of different products ordered**

```sql
SELECT COUNT(DISTINCT P_CODE)
FROM LINE
```
• Find the total number of units ordered

```
SELECT SUM(LINE_UNITS) FROM LINE
```

• Display the invoice number and the total number of units ordered per invoice
• Display the invoice number and the total number of units ordered per invoice

• SELECT INV_NUMBER, SUM(LINE_UNITS)

• FROM LINE

• GROUP BY INV_NUMBER

• Display the invoice number and the total number of units ordered per invoice for customer code ‘112233’
Display the invoice number and the total number of units ordered per invoice for customer code ‘112233’

```
SELECT LINE.INV_NUMBER, SUM(LINE_UNITS)
FROM LINE, INVOICE
WHERE LINE.INV_NUMBER = INVOICE.INV_NUMBER AND CUS_CODE='112233'
GROUP BY LINE.INV_NUMBER
```

Display the invoice number and the total number of units ordered per invoice for customer code ‘112233’ but only for orders exceeding 100$
• Display the invoice number and the total number of units ordered per invoice for customer code ‘112233’ but only for orders exceeding 100$.

```sql
SELECT LINE.INV_NUMBER, SUM(LINE_UNITS) 
FROM LINE, INVOICE 
WHERE LINE.INV_NUMBER = INVOICE.INV_NUMBER AND 
CUS_CODE='112233'
GROUP BY LINE.INV_NUMBER 
HAVING SUM(LINE_UNITS*LINE_PRICE)> 100
```

• Display the invoice number and the total number of units ordered per invoice for customer code ‘112233’ but only for orders with more than two lines.
• Display the invoice number and the total number of units ordered per invoice for customer code ‘112233’ but only for orders with more than two lines

SELECT LINE.INV_NUMBER, SUM(LINE_UNITS)
FROM LINE, INVOICE
WHERE LINE.INV_NUMBER = INVOICE.INV_NUMBER AND CUS_CODE='112233'
GROUP BY LINE.INV_NUMBER
HAVING COUNT(LINE_NUMBER) > 2
Grouping Data

• Frequency distributions created by GROUP BY clause within SELECT statement

• Syntax:

```sql
SELECT columnlist
FROM tablelist
[WHERE conditionlist]
[GROUP BY columnlist]
[HAVING conditionlist]
[ORDER BY columnlist [ASC | DESC] ] ;
```
Incorrect and correct use of the GROUP BY clause

```
SQL> SELECT U_CODE, P_CODE, P_DESCRIPT, P_PRICE
   2   FROM PRODUCT
   3   GROUP BY U_CODE;

SELECT U_CODE, P_CODE, P_DESCRIPT, P_PRICE
```

```
ERROR at line 1:
ORA-00979: not a GROUP BY expression
```

```
SQL> SELECT U_CODE, COUNT(DISTINCT P_CODE)
   2   FROM PRODUCT
   3   GROUP BY U_CODE;

<table>
<thead>
<tr>
<th>U_CODE</th>
<th>COUNT(DISTINCT P_CODE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>21225</td>
<td>2</td>
</tr>
<tr>
<td>21231</td>
<td>1</td>
</tr>
<tr>
<td>21344</td>
<td>3</td>
</tr>
<tr>
<td>23119</td>
<td>2</td>
</tr>
<tr>
<td>24288</td>
<td>3</td>
</tr>
<tr>
<td>25595</td>
<td>2</td>
</tr>
</tbody>
</table>

7 rows selected.
```

SOURCE: Course Technology/Cengage Learning
Joining Database Tables

• Joining tables is the most important distinction between relational database and other DBs
• Join is performed when data are retrieved from more than one table at a time
  – Equality comparison between foreign key and primary key of related tables
• Join tables by listing tables in FROM clause of SELECT statement
  – DBMS creates Cartesian product of every table
Joining Tables with an Alias

- Alias identifies the source table from which data are taken
- Alias can be used to identify source table
- Any legal table name can be used as alias
- Add alias after table name in FROM clause
  - FROM tablename alias
Recursive Joins

• Alias is especially useful when a table must be joined to itself
  – Recursive query
  – Use aliases to differentiate the table from itself
Virtual Tables: Creating a View

- **View**
  - Virtual table based on a SELECT query
- Base tables
- Tables on which the view is based
- **CREATE VIEW** viewname **AS** **SELECT** query
- Typically used for access control or to conveniently access complex queries
• Create a view to show customer names and phone numbers (for marketing people)

CREATE VIEW MARKETING AS ( 
    SELECT CUS_LNAME, CUS_FNAME, CUS_AREAACODE, CUS_PHONE 
    FROM CUSTOMER); 

Select * from MARKETING;
Inserting Table Rows with a SELECT Subquery

• INSERT
  – Inserts multiple rows from another table (source)
  – Uses SELECT subquery
  – Subquery: query embedded (or nested or inner) inside another query
  – Subquery executed first
  – Syntax:

    ```sql
    INSERT INTO tablename (SELECT columnlist FROM tablename WHERE ...);
    ```
Updating Table Rows

- **UPDATE**
  - Modify data in a table
  - Syntax:
    ```
    UPDATE tablename
    SET columnname = expression [, columnname = expression]
    [WHERE conditionlist];
    ```
- If more than one attribute is to be updated in row, separate corrections with commas
- Applies to one table at a time
• Double the line price for product code ‘112233’

```
UPDATE LINE
SET LINE_PRICE = LINE_PRICE*2
WHERE P_CODE = '112233'
```

• Double the line price for product code ‘112233’ for invoices made on ’31-Dec-2011’
Double the line price for product code ‘112233’ for invoices made on ’31-Dec-2011’

```
UPDATE LINE
SET LINE_PRICE = LINE_PRICE*2
WHERE P_CODE = '112233' AND INV_NUMBER IN (SELECT INV_NUMBER
FROM INVOICE
WHERE INV_DATE='31-Dec-2011')
```
Deleting Table Rows

• **DELETE**
  – Deletes a table row
  – Syntax:
    ```sql
    DELETE FROM tablename
    [WHERE conditionlist];
    ```

• WHERE condition is optional

• If WHERE condition is not specified, all rows from specified table will be deleted

• **Applies to one table at a time**
Saving Table Changes & Restoring Table Contents

• Changes made to table contents are not physically saved on disk until:
  – Database is closed
  – Program is closed
  – **COMMIT** command is used

• Will permanently save any changes made to any table in the database

• **ROLLBACK**
  – Undoes changes since last COMMIT
  – Brings data back to pre-change values

• COMMIT and ROLLBACK only work with commands to add, modify, or delete table rows
Advanced Data Definition Commands

• All changes in table structure are made by using \textbf{ALTER} command

• Three options:
  – \textbf{ADD} adds a column or constraint
  – \textbf{MODIFY} changes column characteristics
  – \textbf{DROP} deletes a column or constraint
Deleting a Table from the Database

- **DROP**
  - Deletes table from database
  - Syntax:
    - **DROP TABLE** tablenname;
- Can drop a table only if it is not the “one” side of any relationship
  - Otherwise, RDBMS generates an error message
  - Foreign key integrity violation
Summary

• SQL commands can be divided into two overall categories:
  – Data definition language commands
  – Data manipulation language commands
• The ANSI standard data types are supported by all RDBMS vendors in different ways
• Basic data definition commands allow you to create tables and indexes
Summary (cont’d.)

• DML commands allow you to add, modify, and delete rows from tables
• The basic DML commands:
  – SELECT, INSERT, UPDATE, DELETE, COMMIT, and ROLLBACK
• SELECT statement is main data retrieval command in SQL
Summary (cont’d.)

• WHERE clause can be used with SELECT, UPDATE, and DELETE statements

• Aggregate functions
  – Special functions that perform arithmetic computations over a set of rows

• ORDER BY clause
  – Used to sort output of SELECT statement
  – Can sort by one or more columns
  – Ascending or descending order
Summary (cont’d.)

• Join output of multiple tables with SELECT statement
  – Join performed every time you specify two or more tables in FROM clause
  – If no join condition is specified, DBMX performs Cartesian product
• Natural join uses join condition to match only rows with equal values in specified columns