INTRODUCTION TO STRUCTURED QUERY LANGUAGE (SQL)

ISM 318
Database Systems
Dr. Hamid Nemati

Structured Query Language (SQL)

- Understand basics of database languages
- Introduction to SQL
- Learn how to create a database using SQL
- Learn how to manipulate and manage a database using SQL
- Learn how to use Nested Queries
- Learn how to use Multiple-Table Queries
- Learn how to create VIEWS

Introduction to SQL

- SQL meets ideal database language requirements:
  - SQL coverage fits into three categories:
    1. Data definition:
      - Commands to create the database and its table structures
    2. Data management:
      - Commands used to enter, correct, delete and update data within the database table
    3. Data query:
      - Commands to explore the database contents.
  - SQL is relatively easy to learn.
  - ANSI prescribes a standard SQL.
Introduction to SQL

- Reasons for Studying SQL:
  - The ANSI standardization effort has led to a de facto query standard for relational databases.
  - SQL has become the basis for present and expected future DBMS integration efforts.
  - SQL has become the catalyst in the development of distributed databases and database client/server architecture.

Data Definition Commands

- The Database Model
  - Database ↔ PRODUCT and VENDOR tables
    - Each product is supplied by only a single vendor.
    - A vendor may supply many products.

- Some observations on the PRODUCT and VENDOR tables:
  - The VENDOR table contains vendors who are not referenced in the PRODUCT table. PRODUCT is optional to VENDOR.
  - All V_CODE values in the PRODUCT table must have a match in the vendor table.
  - A few products are supplied factory-direct, a few are made in-house, and a few may have been bought in a special warehouse sale. That is, a product is not necessarily supplied by a vendor. VENDOR is optional to PRODUCT.
### Data Definition Commands

**Creating Table Structures**

```sql
CREATE TABLE <table name>
(<attribute1 name and attribute1 characteristics, attribute2 name and attribute2 characteristics, attribute3 name and attribute3 characteristics, primary key designation, foreign key designation and foreign key requirement>);
```

**Rules for table and attribute names**
- No more than 31 chars long.
- Contain letters (upper or lower case), digits, dollar signs and underscores only.
- End with a letter or digit.

### Data Definition Commands

**Some Common SQL Data Types**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>NUMERIC(l,d)</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>DECIMAL(l,d)</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>CHARACTER(l)</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
</tr>
</tbody>
</table>

### Data Definition Commands

```sql
CREATE TABLE VENDOR
(V_CODE FCHAR(5) NOT NULL UNIQUE,
V_NAME VCHAR(35) NOT NULL,
V_CONTACT VCHAR(15) NOT NULL,
V_AREACODE FCHAR(3) NOT NULL,
V_PHONE FCHAR(3) NOT NULL,
V_STATE FCHAR(2) NOT NULL,
V_ORDER FCHAR(1) NOT NULL,
PRIMARY KEY (V_CODE));
```
Data Definition Commands

CREATE TABLE PRODUCT
(P_CODE VARCHAR(10) NOT NULL UNIQUE,
P_DESCRIPT VARCHAR(35) NOT NULL,
P_INDATE DATE NOT NULL,
P_ONHAND SMALLINT NOT NULL,
P_MIN SMALLINT NOT NULL,
P_PRICE DECIMAL(8,2) NOT NULL,
P_DISCOUNT DECIMAL(4,1) NOT NULL,
V_CODE SMALLINT,
PRIMARY KEY (P_CODE),
FOREIGN KEY (V_CODE) REFERENCES VENDOR
ON DELETE RESTRICT
ON UPDATE CASCADE);

SQL Integrity Constraints
- Entity Integrity
  - PRIMARY KEY
    - NOT NULL and UNIQUE
- Referential Integrity
  - FOREIGN KEY
  - ON DELETE
  - ON UPDATE

Basic Data Management
- Data Entry
  INSERT INTO <table name> VALUES (attribute 1 value,
  attribute 2 value, ... etc.);
- Examples:
  INSERT INTO customers
  VALUES('c001', 'TipTop', 'Duluth' 10.00);
  INSERT INTO Orders
  VALUES(1011, 'jan', 'c001', 'a01', 'p01',
  1000, 450.00);
Basic Data Management

- Checking the Table Contents
  SELECT <attribute names> FROM <table names>;
  - Examples:
    SELECT * FROM products;
    SELECT pid, pname, quantity, price FROM customers;
    SELECT pid, pname, quantity*price FROM customers;

Basic Data Management

- Saving the Table Contents
  COMMIT <table names>;
  - Example:
    COMMIT PRODUCT;

Basic Data Management

- Adding Data to the Table
  INSERT INTO <table name> VALUES(attribute values);
  - Example:
    INSERT INTO PRODUCT
    VALUES('14-Q1/L3', '9.00-in. Per. saw Line', '11/12/96', 18, 12, 17.49, 0.00, 21344);
Basic Data Management

- Making a Correction
  
  \[
  \text{UPDATE} \ <\text{table name}> \\
  \text{SET} \ <\text{attribute name}> = <\text{attribute value}> \\
  \text{WHERE} \ <\text{attribute name}> = <\text{attribute value}>
  \]

  \text{Examples:}
  
  \begin{align*}
  \text{UPDATE PRODUCT} \\
  \text{SET} \ P\text{\_INDATE} & = '12/11/96' \\
  \text{WHERE} \ P\text{\_CODE} & = '13-Q2/P2'; \\
  \text{UPDATE PRODUCT} \\
  \text{SET} \ P\text{\_INDATE} & = '12/11/96', \ P\text{\_PRICE} = 15.99, \\
  \text{P\_MIN} & = 50 \\
  \text{WHERE} \ P\text{\_CODE} & = '13-Q2/P2';
  \end{align*}

- Restoring the Table Contents
  
  \text{ROLLBACK}

Basic Data Management

- Deleting Table Rows
  
  \[
  \text{DELETE FROM} \ <\text{table name}> \\
  \text{WHERE} \ <\text{attribute name}> = <\text{attribute value}>
  \]

  \text{Example:}
  
  \begin{align*}
  \text{DELETE FROM PRODUCT} \\
  \text{WHERE} \ P\text{\_CODE} & = '2289/QPD'; \\
  \text{DELETE FROM PRODUCT} \\
  \text{WHERE} \ P\text{\_MIN} & = 5;
  \end{align*}

Queries

- Partial Listing of Table Contents
  
  \[
  \text{SELECT} \ <\text{column(s)}> \\
  \text{FROM} \ <\text{table name}> \\
  \text{WHERE} \ <\text{conditions}>
  \]

  \text{Examples:}
  
  \begin{align*}
  \text{SELECT} & \ P\text{\_DESCRIPT}, \ P\text{\_INDATE}, \ P\text{\_PRICE}, \ V\text{\_CODE} \\
  \text{FROM} \ \text{PRODUCT} \\
  \text{WHERE} \ V\text{\_CODE} & = 21344;
  \end{align*}

  \begin{center}
  \begin{tabular}{|c|c|c|c|}
  \hline
  \text{P\_DESCRIPT} & \text{P\_INDATE} & \text{P\_PRICE} & \text{V\_CODE} \\
  \hline
  Fire-hydrant (1 1/2 in. hose) & 11/11/96 & $4.95 & 21344 \\
  Fire-hydrant (1 3/4 in. hose) & 6/14/96 & $4.95 & 21344 \\
  3.00 in. pw. saw blade & 11/12/96 & $17.49 & 21344 \\
  \hline
  \end{tabular}
  \end{center}

  \text{Selected \text{PRODUCT} Table Attributes for the VENDOR CODE 21344}
To Retrieve a Record And/or Attribute Value

- The most general form of the SELECT is:

```
SELECT [DISTINCT] attrib-1 [, attrib-2] ...
FROM    table-1 [, table-2]...
[WHERE         condition-1 [ boolean -1 condition-2]...]
[GROUP BY       attrib-3 [ attrib-4]...]
[HAVING        condition-3 [ boolean -1 condition-4]...]
[ORDER BY       attrib-5 [ attrib-6]...]  [ASC|DESC]
```

NOTES:

- The ORDER clause can be used to specify the vertical sequencing of values in the resulting table. Sequencing options include:
  - ASC (for ascending),
  - DESC (for descending) with asc being the default.

- EXAMPLE:
  ```
  SELECT S_NO, STATUS
  FROM S
  WHERE CITY = 'Paris'
  ORDER BY STATUS DESC
  ```

NOTES:

- Select in SQL does not eliminate duplicate rows from the result of the select statement unless user explicitly requests the distinct option.
- Here SQL implementation is at odds with relational theory.
- EXAMPLE:
  ```
  SELECT CITY
  FROM Supplier ;
  ```
  To element duplicates use DISTINCT clause:
  ```
  SELECT DISTINCT CITY
  FROM Supplier ;
  ```
NOTES

- A qualified attribute name consists of a table name and an attribute name, in that order, separated by a period.
- It is never wrong to use qualified name, and sometimes it is essential; e.g., if 2 tables have the same attribute name.
- EXAMPLE:
  
  ```sql
  SELECT S.S_NO, S.STATUS
  FROM Supplier S
  WHERE S.CITY = 'PARIS';
  ```

NOTES

- The SELECT, WHERE and HAVING clauses can include general scalar expressions (involving addition, subtraction, multiplication and division) with attributes, thus providing retrieval of computed values.
- In this case the original attribute name will not be printed above the computed value.
- EXAMPLE:
  
  ```sql
  SELECT P.P_NO, P.WT * 454
  FROM Products P;
  ```

NOTES

- A copy of the entire table can be retrieved by specifying an asterisk instead of attribute(s) in the select clause.
- Example:
  
  ```sql
  SELECT *
  FROM Supplier;
  ```
- You can also select columns from multiple tables:
  
  ```sql
  SELECT P.DESCRIPT, P.PRICE, V_NAME, V_CONTACT, V_AREACODE, V_PHONE
  FROM PRODUCT P, VENDOR V
  WHERE P.V_CODE = V.V_CODE
  ORDER BY P.PRICE;
  ```
### Queries using SELECT

**SELECT** `P_DESCRIPT, P_INDATE, P_PRICE, V_CODE`  
**FROM** `PRODUCT`  
**WHERE** `V_CODE` <> 21344;

<table>
<thead>
<tr>
<th>P_CODE</th>
<th>P_DESCRIPT</th>
<th>P_INDATE</th>
<th>P_PRICE</th>
<th>V_CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>9/2/96</td>
<td>Heat chain saw, 10 ft</td>
<td>7/6/96</td>
<td>256.99</td>
<td>24288</td>
</tr>
<tr>
<td>9/23/96</td>
<td>BID iglum. hole blade</td>
<td>9/29/96</td>
<td>553.67</td>
<td>24288</td>
</tr>
<tr>
<td>8/1/96</td>
<td>Hot cloth, 1/4 in. x 250 ft</td>
<td>8/14/96</td>
<td>28.95</td>
<td>23119</td>
</tr>
<tr>
<td>11/19/95</td>
<td>Claw hammer</td>
<td>11/19/95</td>
<td>5.95</td>
<td>21225</td>
</tr>
<tr>
<td>10/29/96</td>
<td>BID iglum. hole blade</td>
<td>10/29/96</td>
<td>109.92</td>
<td>24288</td>
</tr>
<tr>
<td>8/14/96</td>
<td>Hot cloth, 1/2 in. x 350 ft</td>
<td>8/14/96</td>
<td>43.99</td>
<td>23119</td>
</tr>
<tr>
<td>10/19/96</td>
<td>BID iglum. hole blade</td>
<td>10/19/96</td>
<td>43.99</td>
<td>23119</td>
</tr>
<tr>
<td>11/16/96</td>
<td>Steel nail, 4d by 1 1/2 in, 5 5/8 mesh</td>
<td>11/16/96</td>
<td>119.95</td>
<td>29999</td>
</tr>
<tr>
<td>1/2/96</td>
<td>Power pumps, 15 ps. 2 speed</td>
<td>1/2/96</td>
<td>109.93</td>
<td>29999</td>
</tr>
</tbody>
</table>

**Selected PRODUCT Table Attributes for VENDOR CODE Other Than 21344**

### Queries using SELECT

**SELECT** `P_DESCRIPT, P_ONHAND, P_MIN, P_PRICE`  
**FROM** `PRODUCT`  
**WHERE** `P_PRICE` <= 10;

<table>
<thead>
<tr>
<th>P_CODE</th>
<th>P_DESCRIPT</th>
<th>P_ONHAND</th>
<th>P_MIN</th>
<th>P_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>8 in. by 10 ft</td>
<td>123</td>
<td>100</td>
<td>9.95</td>
</tr>
<tr>
<td>456</td>
<td>4 in. by 10 ft</td>
<td>456</td>
<td>100</td>
<td>9.95</td>
</tr>
<tr>
<td>789</td>
<td>Riveted flange, 1/16 in.</td>
<td>789</td>
<td>30</td>
<td>4.99</td>
</tr>
<tr>
<td>123</td>
<td>Claw hammer</td>
<td>123</td>
<td>10</td>
<td>3.99</td>
</tr>
<tr>
<td>245</td>
<td>1 25 ft. steel core, 5 ft</td>
<td>245</td>
<td>75</td>
<td>6.95</td>
</tr>
</tbody>
</table>

**Selected PRODUCT Table Attributes with a P-PRICE Restriction**

### Queries using mathematical operators

**Using Mathematical Operators on Character Attributes**

**SELECT** `P_DESCRIPT, P_ONHAND, P_MIN, P_PRICE`  
**FROM** `PRODUCT`  
**WHERE** `P_CODE` < 'T558-QW1';

<table>
<thead>
<tr>
<th>P_CODE</th>
<th>P_DESCRIPT</th>
<th>P_ONHAND</th>
<th>P_MIN</th>
<th>P_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>12346</td>
<td>7 1/2 in. car saw blade</td>
<td>12346</td>
<td>15</td>
<td>14.99</td>
</tr>
<tr>
<td>12345</td>
<td>Hot cloth, 1/4 in. x 500 ft</td>
<td>12345</td>
<td>8</td>
<td>35.95</td>
</tr>
<tr>
<td>12344</td>
<td>5 1/2 in. car saw blade</td>
<td>12344</td>
<td>12</td>
<td>17.43</td>
</tr>
<tr>
<td>23456</td>
<td>Power pumps, 15 ps. 3 speed</td>
<td>23456</td>
<td>5</td>
<td>109.93</td>
</tr>
</tbody>
</table>

**Selected PRODUCT Table Attributes: The ASCII Code Effect**
Queries on Dates

Using Mathematical Operators on Dates

```
SELECT P_DESCRIPT, P_ONHAND, P_MIN, P_PRICE
FROM PRODUCT
WHERE P_INDATE >= '11/25/96';
```

<table>
<thead>
<tr>
<th>P_DESCRIPT</th>
<th>P_ONHAND</th>
<th>P_MIN</th>
<th>P_PRICE</th>
<th>P_INDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sledge hammer, 12&quot;</td>
<td>8</td>
<td>5</td>
<td>14.40</td>
<td>12/1/96</td>
</tr>
<tr>
<td>PVC pipe, 3/4&quot; x 8'</td>
<td>188</td>
<td>75</td>
<td>5.00</td>
<td>12/15/96</td>
</tr>
<tr>
<td>1.25 in. metal screw, 25</td>
<td>172</td>
<td>75</td>
<td>6.95</td>
<td>11/25/96</td>
</tr>
<tr>
<td>Power painter, 15 pc, 3 nozzles</td>
<td>8</td>
<td>5</td>
<td>106.98</td>
<td>12/2/96</td>
</tr>
</tbody>
</table>

Selected PRODUCT Table Attributes:

- Date Restriction

Queries using logical operators

- Logical Operators: AND, OR, and NOT
- Examples:

```
SELECT P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM PRODUCT
WHERE V_CODE = 21344
OR V_CODE = 24288;
```

```
SELECT P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM PRODUCT
WHERE P_PRICE < '50'
AND P_INDATE > '07/15/96';
```

```
SELECT * FROM PRODUCT
WHERE V_CODE NOT 21344;
```

```
SELECT S_NO
FROM SUPPLIER
WHERE CITY = 'PARIS'
AND STATUS > 20;
```

Examples Queries using logical operators

```
SELECT P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM PRODUCT
WHERE D_PRICE < 50
AND P_INDATE > 07/15/96;
```

```
SELECT * FROM PRODUCT
WHERE V_CODE NOT 21344;
```

```
SELECT S_NO
FROM SUPPLIER
WHERE CITY = 'PARIS'
AND STATUS > 20;
```
Queries using COMPARISON operators

- The condition following WHERE may include the COMPARISON operators =, <>, >, >=, <, and <=; the boolean operators AND, OR, NOT; and Parentheses to indicate a desired order of evaluation.

```
SELECT P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM PRODUCT
WHERE (P_PRICE < 50 AND P_INDATE > '07/15/96')
OR V_CODE = 24288;
```

Queries using BETWEEN

- Special Operators
  - BETWEEN is used to define range limits.
  - Example:
    ```
    SELECT *
    FROM PRODUCT
    WHERE P_PRICE BETWEEN 50.00 AND 100.00;
    ```

Queries

- IS NULL is used to check whether an attribute value is null.
  - Examples:
    ```
    SELECT P_CODE, P_DESCRIPT
    FROM PRODUCT
    WHERE P_MIN IS NULL;
    ```
    ```
    SELECT P_CODE, P_DESCRIPT
    FROM PRODUCT
    WHERE P_INDATE IS NULL;
    ```
Queries using LIKE condition

LIKE is used to check for similar character strings.
It is similar to the WILD CARD.

- **Examples:**
  - List all V_CONTACTs whose names begin with Smith:
    
    ```
    SELECT * FROM VENDOR
    WHERE V_CONTACT LIKE 'Smith';
    ```
  - List all V_CONTACTs whose names does NOT contain letter S:
    
    ```
    SELECT * FROM VENDOR
    WHERE V_CONTACT NOT LIKE 'S%';
    ```

Structured Query Language (SQL)

```sql
SELECT * FROM VENDOR
WHERE UPPER(V_CONTACT) LIKE 'SMITH%';
```

```sql
SELECT * FROM VENDOR
WHERE V_CONTACT NOT LIKE 'Smith%';
```

Selected PRODUCT Table Attributes:
Partial String Comparisons

```sql
SELECT * FROM PRODUCT
WHERE V_CONTACT LIKE 'Johns_n';
```

Queries

```sql
SELECT * FROM VENDOR
WHERE V_CODE IN (21344, 24288);
```

```sql
DELETE FROM PRODUCT
WHERE P_CODE EXISTS;
```

```sql
SELECT * FROM PRODUCT
WHERE V_CODE EXISTS;
```

Queries using IN Clause

IN is used to check whether an attribute value matches a value contained within a (sub)set of listed values.

**Purpose:** the word "in" furnishes a concise way of asking whether a specified value is contained in a set, or list, of values; used in a where clause.

- **Example:**
  
  ```
  SELECT * FROM PRODUCT
  WHERE V_CODE IN (21344, 24288);
  ```

**EXISTS** is used to check whether an attribute has value.

- **Example:**
  
  ```
  SELECT * FROM PRODUCT
  WHERE P_CODE EXISTS;
  ```
Structured Query Language (SQL)

Advanced Data Management Commands

● Changing Table Structures

ALTER TABLE <table name>
MODIFY <column name> <new column characteristics>;

ALTER TABLE <table name>
ADD <column name> <new column characteristics>;

● Changing a Column's Data Type

Example:
ALTER TABLE PRODUCT
MODIFY (V_CODE CHAR(5));

● Changing Attribute Characteristics

Example:
ALTER TABLE PRODUCT
MODIFY (P_PRICE DECIMAL(8,2));

● Adding a New Column to the Table

Example:
ALTER TABLE PRODUCT
ADD (P_SALECODE CHAR(1));

● Entering Data into the New Column

UPDATE <table name>
SET <attribute name> = <attribute value>
WHERE <condition>

Examples:
UPDATE PRODUCT
SET P_SALECODE = '2'
WHERE P_CODE = '1546-QQ2';

Selected PRODUCT Table Attributes:
Effect of Data Entry into the New P_SALECODE Attribute
Structured Query Language (SQL)

Advanced Data Management Commands

**UPDATE PRODUCT**

```
SET P_SALECODE = '1'
WHERE P_CODE IN ('2232/QWE', '2232/QTY');
```

<table>
<thead>
<tr>
<th>P_CODE</th>
<th>P_SALECODE</th>
<th>P_INDATE</th>
<th>P_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2232/QWE</td>
<td>000</td>
<td>12/15/95</td>
<td>$19.99</td>
</tr>
<tr>
<td>2232/QWE</td>
<td>002</td>
<td>12/15/96</td>
<td>$20.99</td>
</tr>
</tbody>
</table>

Selected PRODUCT Table Attributes:
- Multiple Data Entry

**UPDATE PRODUCT**

```
SET P_SALECODE = '2'
WHERE P_INDATE < 8/15/96;
```

```
SET P_SALECODE = '1'
WHERE P_INDATE >= '11/15/96'
AND P_INDATE < '12/1/96';
```

**Copying Parts of Tables**

```
INSERT INTO <receiving table> (receiving table's column names)
SELECT (column names of the columns to be copied)
FROM <contributing table name>;
```

**Example:**

```
INSERT INTO PART
(PART_CODE, PART_DESCRIPT, PART_PRICE)
SELECT P_CODE, P_DESCRIPT, P_PRICE
FROM PRODUCT;
```
Advanced Data Management Commands

- Deleting a Table from the Database
  DROP TABLE <table name>;
- Example:
  DROP TABLE PART;

Advanced Data Management Commands

- Primary and Foreign Key Designation
  - Examples:
    ALTER TABLE PRODUCT
    ADD PRIMARY KEY (P_CODE);
    ALTER TABLE PRODUCT
    ADD FOREIGN KEY (V_CODE) REFERENCES VENDOR;

More Complex Queries and SQL Functions

- Ordering a Listing
  ORDER BY <attributes>
- Examples:
  SELECT P_CODE, P_DESCRIP, P_INDATE, P_PRICE
  FROM PRODUCT
  ORDER BY P_PRICE;
More Complex Queries and SQL Functions

```
SELECT P_CODE, P_DESCRIPT, P_INDATE, P_PRICE
FROM PRODUCT
WHERE P_INDATE < '9/15/96'
AND P_PRICE <= 50.00
ORDER BY V_CODE, P_PRICE, DESC;
```

A Query Based on Multiple Restriction
More Complex Queries and SQL Functions

- Listing Unique Values
  
  \[ \text{SELECT DISTINCT <attributes> ...} \]
  
  Example:
  
  \[
  \text{SELECT DISTINCT V\_CODE} \\
  \text{FROM PRODUCT;}
  \]

  A Listing of Distinct (Different) V\_CODE Values in the PRODUCT Table

Aggregate Functions Used In The Select Clause

- PURPOSE: To provide a way to summarize the collection of values in one column of a table.

- General format:
  
  \[
  \text{SELECT aggreg-fcn ([DISTINCT ] attrib-name)} \\
  \text{FROM table-name ;}
  \]

- aggreg-fcn Is One of the Following:
  
  - COUNT -- NUMBER OF ROWS IN attrib-name
  - SUM -- SUM OF THE VALUES IN attrib-name
  - AVG -- AVERAGE OF THE VALUES IN attrib-name
  - MAX -- LARGEST VALUE IN attrib-name
  - MIN -- SMALLEST VALUE IN attrib-name

More Complex Queries and SQL Functions

- SQL's Numeric Functions
  
  COUNT
  
  Example:
  
  \[
  \text{SELECT COUNT(DISTINCT V\_CODE)} \\
  \text{FROM PRODUCT;}
  \]
  
  \[
  \text{SELECT COUNT(DISTINCT V\_CODE)} \\
  \text{FROM PRODUCT} \\
  \text{WHERE P\_PRICE <= 10.00;}
  \]
More Complex Queries and SQL Functions

MAX and MIN
- Examples:
  ```sql
  SELECT MAX(P_PRICE)
  FROM PRODUCT;
  
  SELECT MIN(P_PRICE)
  FROM PRODUCT;
  
  SELECT P_CODE, P_DESCRIPT, P_PRICE
  FROM PRODUCT
  WHERE P_PRICE =
    (SELECT MAX(P_PRICE) FROM PRODUCT);
  ```

AVG
- Example:
  ```sql
  SELECT P_DESCRIPT, P_ONHAND, P_PRICE, V_CODE
  FROM PRODUCT
  WHERE P_PRICE >
    (SELECT AVG(P_PRICE) FROM PRODUCT)
  ORDER BY P_PRICE DESC;
  ```

SUM
- Example:
  ```sql
  SELECT SUM(P_ONHAND * P_PRICE)
  FROM PRODUCT;
  ```
NOTES ON AGGREGATE FUNCTIONS:

- Distinct causes duplicate values to be eliminated before the function is applied; not used with max or min.
- For sum and avg, the column must contain numeric values.
- For count, distinct must be specified; the special function count(*) is provided to count all rows in a table without any duplicate elimination.
- To label your output, include a literal in quotes as an attribute:

  ```sql
  SELECT 'TOTAL # SHIPMENTS =', COUNT(*)
  FROM SUPPLIER
  WHERE P_NO = 'P2';
  ```

GROUP BY Clause

- Purpose: the GROUP BY operator logically rearranges the table represented by the FROM clause into partitions or groups, such that within any one group all rows have the same value for the group by field. The SELECT clause is then applied to each group of the partitioned table (instead of to each row of the original table).
- When rows are grouped, one line of output is produced for each group.
- GROUP BY attrib-name [, attrib-name]...
- Examples:

  ```sql
  SELECT P_SALECODE, MIN(P_PRICE)
  FROM PRODUCT_2
  GROUP BY P_SALECODE;
  ```

  ```sql
  SELECT P_SALECODE, AVG(P_PRICE)
  FROM PRODUCT_2
  GROUP BY P_SALECODE;
  ```

Examples Grouping Data

- Compute the total quantity supplied for each part in the sp table.

  ```sql
  SELECT P_NO, SUM(QTY)
  FROM SP
  GROUP BY P_NO;
  ```

- Note: the steps, and their order of execution, in the above example are:
  - Get (and join) the table(s) in the FROM clause.
  - Group the table by the group attrib-name(s).
  - Apply the select clause to each group of attrib-name(s), resulting in one tuple for each group.
- Note: the only attributes which can be specified in the select clause are the attributes used to group the tuples and the aggregate statistic.
HAVING CLAUSE

- Purpose: the HAVING clause is used to eliminate groups just as the WHERE clause is used to eliminate rows.
- The HAVING clause does for groups what the WHERE does for rows.
- Example:
  
  ```sql
  SELECT CREDIT_LIMIT, COUNT(*)
  FROM CUSTOMER
  GROUP BY CREDIT_LIMIT
  HAVING COUNT(*) > 2;
  ```

More Complex Queries and SQL Functions

- SQL Indexes
  ```sql
  CREATE INDEX <index name>
  ON <table name>(<attribute name>);
  ```
- Examples:
  ```sql
  CREATE INDEX P_CODEX
  ON PRODUCT(P_CODE);
  CREATE UNIQUE INDEX P_CODEX
  ON PRODUCT(P_CODE);
  ```

Complex Queries Involving Two or More Tables

- Purpose: the ability to join two or more tables is one of the most powerful features of relational systems.
- The tables to be joined are specified in the "from" clause and the conditions of the join are specified in the "where" clause of the select statement.
- Example:
  ```sql
  SELECT S_NO, SNAME, S.CITY, P_NO, PNAME
  FROM SUPPLIER S, PART P
  WHERE S.CITY = P.CITY;
  ```
Examples of Multiple-Table Queries

```
SELECT P_DESCRIP, P_PRICE, V_NAME, V_CONTACT, V_AREA_CODE, V_PHONE
FROM PRODUCT, VENDOR
WHERE PRODUCT.V_CODE = VENDOR.V_CODE
AND P_INDATE > '11/15/96';

SELECT PRODUCT.P_DESCRIP, PRODUCT.P_PRICE, VENDOR.V_NAME, VENDOR.V_CONTACT, VENDOR.V_AREA_CODE, VENDOR.V_PHONE
FROM PRODUCT, VENDOR
WHERE PRODUCT.V_CODE = VENDOR.V_CODE;

SELECT P_DESCRIP, P_PRICE, V_NAME, V_CONTACT, V_AREA_CODE, V_PHONE
FROM PRODUCT, VENDOR
WHERE PRODUCT.V_CODE = VENDOR.V_CODE
ORDER BY P_PRICE;
```

Another Example of Multiple-Table Queries

- Get all combinations of suppliers and parts where the supplier and part concerned are located in the same city, omitting suppliers with status of 20.

```
SELECT S_NAME, STATUS, S.CITY, PNAME
FROM SUPPLIER S, PART P
WHERE S.CITY = P.CITY
AND STATUS <> 20;

RESULTS:

S_NAME     STATUS  S.CITY   PNAME
JONES 10    PARIS    BOLT
JONES 10    PARIS    CAM
BLAKE 30    PARIS    BOLT
BLAKE 30    PARIS    CAM
```

Another Example of Multiple-Table Queries

- For each PC, list the tag number and computer id together with the number and name of the employee to whom the PC has been assigned.

```
SELECT TAGNUM, COMPID, EMPLOYEE.EMPNUM, EMPNAME
FROM PC, EMPLOYEE
WHERE PC.EMPNUM = EMPLOYEE.EMPNUM;
```
Another Example of Multiple-Table Queries

- For each PC whose location is "home", list the tag number and computer id together with the number and name of the employee to whom the PC has been assigned.

```
SELECT TAGNUM, COMPID, EMPLOYEE.EMPNUM, EMPNAME
FROM PC, EMPLOYEE
WHERE PC.EMPNUM = EMPLOYEE.EMPNUM
AND LOCATION = 'HOME';
```

Another Example of Multiple-Table Queries

- For each package that has been installed on a PC, find the tag number of the PC, the package id, and the package name.

```
SELECT TAGNUM, SOFTWARE.PACKID, PACKNAME
FROM SOFTWARE, PACKAGE
WHERE SOFTWARE.PACKID = PACKAGE.PACKID;
```

Note Of Multiple-table Queries

- Note:
  - Whenever there is potential ambiguity in the attribute names, we must qualify the attribute name(s) involved; e.g., the attribute "city" appears in both s and p tables, so we must qualify it by specifying the table-name and the attrib-name separated by a period.
  - Thus, we specify s.city and p.city.
Nested Queries

- **Purpose:** It is possible to place one query inside another. The inner query is called a subquery and is executed first and its result is then used in the outer query.

- **EXAMPLE:**
  
  Get supplier numbers for suppliers who are located in the same city as supplier s1.
  
  ```
  SELECT S_NO
  FROM SUPPLIER S
  WHERE CITY =
  (SELECT CITY
   FROM S
   WHERE S_NO = 'S1');
  ```

Another Example of Nested Queries

- **List the part number and name for all parts whose weight is greater than the average weight of all screws.**

  ```
  SELECT P_NO, PNAME
  FROM PART P
  WHERE WEIGHT >
  (SELECT AVG(WEIGHT)
   FROM P
   WHERE PNAME = 'SCREW');
  ```

VIEWS (Virtual Tables)

- **Definition:** A view is a table whose data is not physically stored, but is rather a virtual structure that refers to rows stored in other tables. Its purpose is to give each user his or her own picture of the database.

- **Format for creating a view:**

  ```
  CREATE VIEW <view name> AS
  SELECT ... FROM ... WHERE ...;
  ```

- **Example:**

  ```
  CREATE VIEW PRODUCT_3 AS
  SELECT P_DESCRIP, P_ONHAND, P_PRICE
  FROM PRODUCT
  WHERE P_PRICE > 50.00;
  ```
Examples of Views

- You can define a view based on more than one base table:

  ```sql
  CREATE VIEW SP_NEW AS
  SELECT SNAME, PNAME, QTY
  FROM S, SP, P
  WHERE S.SNO = SP.SNO
  AND P.PNO = SP.PNO;
  ```

- Once you created a view, you can use like just like a table:

  ```sql
  SELECT *
  FROM SP_NEW
  WHERE SNAME = 'JONES';
  ```

- You can give local names to a view's columns to make them more meaningful to the user:

  ```sql
  CREATE VIEW CLARK (SUPPLIER, PART, QUANTITY) AS
  SELECT SNAME, PNAME, QTY
  FROM S, SP, P
  WHERE S.SNO = SP.SNO
  AND P.PNO = SP.PNO
  AND SNAME = 'CLARK';

  SELECT *
  FROM CLARK;
  ```

- RESULT:

<table>
<thead>
<tr>
<th>SUPPLIER</th>
<th>PART</th>
<th>QUANTITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLARK</td>
<td>BOLT</td>
<td>200</td>
</tr>
<tr>
<td>CLARK</td>
<td>SCREW</td>
<td>300</td>
</tr>
<tr>
<td>CLARK</td>
<td>CAM</td>
<td>400</td>
</tr>
<tr>
<td>CLARK</td>
<td>CAM</td>
<td>500</td>
</tr>
<tr>
<td>CLARK</td>
<td>CAM</td>
<td>500</td>
</tr>
<tr>
<td>CLARK</td>
<td>CAM</td>
<td>100</td>
</tr>
</tbody>
</table>

- You can include statistics in a view:

  ```sql
  CREATE VIEW PART_SUMMARY (PART, TOTAL_QTY) AS
  SELECT PNO, SUM(QTY)
  FROM SP
  GROUP PNO;

  SELECT *
  FROM PART_SUMMARY;
  ```

- RESULT:

<table>
<thead>
<tr>
<th>PART</th>
<th>TOTAL_QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>600</td>
</tr>
<tr>
<td>P2</td>
<td>1000</td>
</tr>
<tr>
<td>P3</td>
<td>400</td>
</tr>
<tr>
<td>P4</td>
<td>500</td>
</tr>
<tr>
<td>P5</td>
<td>500</td>
</tr>
<tr>
<td>P6</td>
<td>100</td>
</tr>
</tbody>
</table>
Examples of Views

- You can define a view with calculated attributes, but you must name the calculated attributes in the view:

  CREATE VIEW METRIC (PNO, PNAME, COLOR, METRIC_WT) AS
  SELECT PNO, PNAME, COLOR, WEIGHT*454/16
  FROM P;

  SELECT *
  FROM METRIC
  WHERE PNAME = 'SCREW';

- RESULTS:

  PNO PNAME COLOR METRIC_WT
  P3 SCREW BLUE 482
  P4 SCREW RED 397

Examples of Views

- You can define a view which is dependent on another view:

  CREATE VIEW RED_PARTS AS
  SELECT *
  FROM METRIC
  WHERE COLOR = 'RED';

  SELECT PNO, PNAME
  FROM RED_PARTS;

- RESULTS:

  PNO PNAME
  P1 NUT
  P4 SCREW
  P6 COG

Notes On Views

- The differences between a view and a base relation is transparent to the user; i.e., A view can be requested, operated on, updated, etc. Just like a base relation.
- However, updating some views is difficult or impossible
- Updating views is limited by the following:
  - a view with a single defining table is updatable if the view attributes contain the primary key or some other candidate key of the base relation.
  - views defined on multiple base tables using joins are generally not updatable.
  - views defined using grouping and aggregate functions are not updatable.
Notes On Views

- Views can provide logical data independence; i.e., Users and programs (in the case of imbedded SQL) are immune to changes in the logical structure of the database, namely:
  - **Database growth** - the expansion of an existing base relations to include a new attribute, or the inclusion of a new base relation or a new object in the database.
  - **Database restructuring** - the changing of the placement of information within the data base; e.g., The allocation of attributes to base relations is altered in some way, such as two tables being joined to form a new base relation.