

## 4. SQL

### 4.1 The language

- SQL is the “structured query language” for relational systems.
- The early SQL standards was published by ANSI and IBM.
- SQL has been universally accepted. Newer SQL standards (SQL89 and SQL92) were published by ISO/IEC. The standards of SQL:1999 were published in 1999, which incorporates the object technology into the relational model.
- SQL can be used both interactively (on-line) from a terminal and embedded in application programs.
- SQL is a complete language. It has its DDL and DML. It can be used for
  - structuring databases
  - retrieving data
  - modifying data

### 4.2 SQL Queries

Query skeleton:

```
SELECT          [ ALL | DISTINCT] column-list | *
FROM           relation-list
[WHERE          condition]
[GROUP BY      column-list
[HAVING condition]]
[ORDER BY     column-list ]
```

- The **SELECT** clause lists the columns desired in the result. An \* represents all the columns.
- **DISTINCT** specifies elimination of duplicates in the result.
- The **FROM** clause lists the relations from which the query is computed.
- The **WHERE** clause specifies the condition the tuples must satisfy to qualify for the result.

- GROUP BY and HAVING are used for aggregation.
- The ORDER BY clause specifies how to sort the output.
- A select-from-where query corresponds closely to a join-select-project expression in relational algebra.

### 4.3 Example Queries

- **Example 1:** (Minimal query)

List all the attributes of all the vendors.

```
SELECT *
FROM v
```

- An “\*” in the SELECT clause indicates all the attributes.
- When there is not a WHERE clause, all the tuples qualify.

- **Example 2:**

Get Vno’s and Vname’s (sorted by Vname) of those vendors in Waterloo having a balance over 300.

```
SELECT Vno, Vname
FROM v
WHERE City = 'Waterloo' AND
       Vbalance > 300
ORDER BY Vname
```

A WHERE clause condition is constructed from

- column names
- constants
- subqueries
- parentheses
- arithmetic operators (+, −, \*, /)

- comparison operators (=, >=, >, <>, <, <=)
- boolean operators (AND, OR, NOT)

• **Example 3:**

Get the account numbers and names of those customers whose name starts with “S”, who live in Ontario or BC, and who have a balance between 2000 and 3500.

```
SELECT  Account, Cname
FROM    c
WHERE   Cname LIKE 'S%' AND
        Province IN ('ONT', 'BC') AND
        Cbalance BETWEEN 2000 AND 3500
```

Additional operators:

- column-name [ NOT ] LIKE pattern  
where pattern may include the wildcard characters ‘%’ and ‘\_’ (underscore). The character ‘%’ matches any string and ‘\_’ matches any (single) character.
- column-name [ NOT ] IN (set of values)  
tests for set membership.
- arithm-exp [ NOT ] BETWEEN const1 AND const2  
tests if something is within a range.

• **Example 4:**

Get the account numbers and names of those customers who have a transaction with a value over 2000.

Solution 1:

```
SELECT  DISTINCT C.Account, Cname
FROM    c, t
WHERE   c.Account=t.Account AND
        t.Amount > 2000
```

- Note that DISTINCT is needed in this solution.
- “c.Account=t.Account” is a join condition.

Solution 2:

```
SELECT  Account, Cname
FROM    c
WHERE   Account IN (SELECT Account
                    FROM    t
                    WHERE   Amount > 500)
```

- This is a query with a subquery.
- When using IN, the subquery must return a set of values of a single column.
- An inner query (subquery) can reference columns of tables in the outer query.
- The outer query cannot reference columns of tables in a subquery.

• **Example 5:**

Get the account numbers of those customers having transactions with both vendors V2 and V3.

Solution 1:

```
SELECT  DISTINCT t1.Account
FROM    t t1, t t2
WHERE   t1.Account = t2.Account AND
        t1.Vno = 'V2' AND
        t2.Vno = 'V3'
```

- t1 and t2 are correlation names.

Solution 2:

```
SELECT  Account
FROM    t
WHERE   Vno = 'V2' AND
        Account IN (SELECT Account
                    FROM    t
                    WHERE   Vno = 'V3')
```

- **Example 6:**

Get the account numbers and names of those customers whose transactions are all over 200.

```
SELECT Account, Cname
FROM c
WHERE 200 <ALL(SELECT Amount
                FROM t
                WHERE c.Account=t.Account)
```

- This query has a subquery.
- forms of ALL:  
<ALL, <=ALL, =ALL, >=ALL, > ALL, <> ALL
- option ALL (set of values)  
evaluates to **true** iff the comparison evaluates to **true** for every value in the set.
- option SOME (set of values)  
evaluates to **true** iff the comparison evaluates to **true** at least one value in the set.

- **Example 7:**

Get names of those vendors who had at least one transaction during September 1999.

```
SELECT Vname
FROM v
WHERE EXISTS (SELECT *
              FROM t
              WHERE Vno = v.Vno AND
                    T_Date BETWEEN
                    990901 AND 990930)
```

- This query has an EXISTS (SELECT ... ) subquery.
- This condition evaluates to **true** iff the resulting set from the subquery contains at least one row.

- **Example 8:**

Count the number of vendors having transactions with at least 5 different customers.

```

SELECT COUNT(*)
FROM v
WHERE 5 <=(SELECT COUNT(DISTINCT Account)
           FROM t
           WHERE Vno = v.Vno)

```

- COUNT is one of several **built-in** or **aggregation** functions provided by SQL.
- Aggregation functions: COUNT, SUM, AVG, MIN, MAX

• **Example 9:**

For each account having more than one transaction, give the account numbers and the total amounts of all the transactions.

```

SELECT Account, SUM(Amount)
FROM t
GROUP BY Account
HAVING COUNT (*) > 1

```

- “GROUP BY Account” partitions the tuples in t into groups according to Account number, i.e. within a group all tuples have the same Account number.
- The HAVING clause is then applied to each group separately, to determine whether the group qualifies.
- Aggregation functions like SUM are applied separately to each group in a partitioned relation.
- All attributes in the SELECT clause to which no aggregation function is applied, must be mentioned in the GROUP BY clause.

#### 4.4 Creating and Deleting Tables

##### CREATE

The CREATE command is used to define a table.

```

CREATE TABLE table-name
(column-name type, column-name type, ...)

```

- **Example 10:**

Create a table containing account number, name, province, balance, and credit limit of customers.

```
CREATE TABLE c
  (Account CHAR(5), Cname CHAR(20),
   Province CHAR(3), Cbalance DECIMAL(2),
   Crlimit INTEGER)
```

## DROP

The DROP command is used to delete the definition of a table.

```
DROP TABLE table-name.
```

## 4.5 Updating Tables

SQL Update Commands: UPDATE, INSERT, DELETE

### UPDATE

The UPDATE command modifies some tuples in a table.

```
UPDATE table-name
SET      column-name = value-expression, ...
[WHERE condition]
```

- **Example 11:**

Double the credit limit and set the balance to zero for all Ontario customers.

```
UPDATE c
SET      Crlimit = Crlimit*2, Cbalance = 0
WHERE   Province = 'ONT'
```

- **Example 12:**

Reduce the credit limit of every Sears customer by 10%.

```
UPDATE c
SET      Crlimit = Crlimit*0.9
WHERE   Account IN (SELECT Account
                    FROM   t, v
                    WHERE  t.Vno = v.Vno AND
                           Vname = 'Sears')
```

## INSERT

The INSERT command adds a new tuple to a table.

```
INSERT INTO table-name
VALUES (value, value, ...)
```

- **Example 13:**

Add a new vendor.

```
INSERT INTO v
VALUES ('V6', 'Zellers', 'Kitchener', 0)
```

## DELETE

The DELETE command removes tuples from a table.

```
DELETE FROM table-name
WHERE condition
```

- **Example 14:**

Delete all customers who have no transactions

```
DELETE FROM c
WHERE Account NOT IN (SELECT Account
                      FROM t)
```

- **Example 15:**

Delete the Waterloo WalMart from the v table.

```
DELETE FROM v
WHERE Vname = 'WalMart' AND City = 'Waterloo'
```

Note that an UPDATE, INSERT or DELETE command modifies **one** table only.