SQL, the Structured Query Language

Overview

- Introduction
- DDL Commands
- DML Commands
- SQL Statements, Operators, Clauses
- Aggregate Functions

Structured Query Language (SQL)

- The ANSI standard language for the definition and manipulation of relational database.

- Includes data definition language (DDL), statements that specify and modify database schemas.

- Includes a data manipulation language (DML), statements that manipulate database content.

Some Facts on SQL

- SQL data is case-sensitive, SQL commands are not.
- First Version was developed at IBM by Donald D. Chamberlin and Raymond F. Boyce. [SQL]
- Developed using Dr. E.F. Codd's paper, “A Relational Model of Data for Large Shared Data Banks.”
- SQL query includes references to tuples variables and the attributes of those variables

SQL: DDL Commands

- CREATE TABLE: used to create a table.
- ALTER TABLE: modifies a table after it was created.
- DROP TABLE: removes a table from a database.

SQL: CREATE TABLE Statement

- Things to consider before you create your table are:
  - The type of data
  - the table name
  - what column(s) will make up the primary key
  - the names of the columns

- CREATE TABLE statement syntax:

```
CREATE TABLE <table name>
( field1 datatype ( NOT NULL ),
  field2 datatype ( NOT NULL )
);
```

SQL: Attributes Types

Numeric types	integer	`integer, int, smallint, long`
	floating point	`float, real, double precision`
	formatted	`decimal(i,j), dec(i,j)`
Character-string types	fixed length	`char(n), character(n)`
	varying length	`varchar(n), char varying(n), character varying(n)`
Bit-string types	fixed length	`bit(n)`
	varying length	`bit varying(n)`
Date and time types		`date, time, datetime, timestamp, time with time zone, interval`
Large types	character	`long varchar(n), clob, text`
	binary	`blob`

SQL: ALTER TABLE Statement

- To add or drop columns on existing tables.

- ALTER TABLE statement syntax:

ALTER TABLE <table name>

ADD attr datatype;

or

DROP COLUMN attr;

SQL: DROP TABLE Statement

Has two options:

- CASCADE: Specifies that any foreign key constraint violations that are caused by dropping the table will cause the corresponding rows of the related table to be deleted.

- RESTRICT: blocks the deletion of the table of any foreign key constraint violations would be created.

DROP TABLE statement syntax:
DROP TABLE <table name> [RESTRICT|CASCADE];

Example:

```
CREATE TABLE FoodCart (
date varchar(10),
food varchar(20),
profit float
);
```

FoodCart

date	food	profit

```
ALTER TABLE FoodCart (
ADD sold int
);
```

FoodCart

date	food	profit	sold

```
ALTER TABLE FoodCart(
DROP COLUMN profit
);
```

FoodCart

date	food	sold

```
DROP TABLE FoodCart;
```

SQL: DML Commands

- INSERT: adds new rows to a table.
- UPDATE: modifies one or more attributes.
- DELETE: deletes one or more rows from a table.

SQL: INSERT Statement

- To insert a row into a table, it is necessary to have a value for each attribute, and order matters.
- INSERT statement syntax:

INSERT into <table name>
VALUES ('value1', 'value2', NULL);

Example: INSERT into FoodCart
VALUES ('02/26/08', 'pizza', 70);

FoodCart

date	food	sold
02/25/08	pizza	350
02/26/08	hotdog	500

date	food	sold
02/25/08	pizza	350
02/26/08	hotdog	500
02/26/08	pizza	70

SQL: UPDATE Statement

- To update the content of the table:

UPDATE statement syntax:

UPDATE <table name> SET <attr> = <value>

WHERE <selection condition>;

Example: UPDATE FoodCart SET sold = 349

WHERE date = '02/25/08' AND food = 'pizza';

FoodCart

date	food	sold
02/25/08	pizza	350
02/26/08	hotdog	500
02/26/08	pizza	70

date	food	sold
02/25/08	pizza	349
02/26/08	hotdog	500
02/26/08	pizza	70

SQL: DELETE Statement

- To delete rows from the table:

DELETE statement syntax:
DELETE FROM <table name>
WHERE <condition>;
Example: DELETE FROM FoodCart
WHERE food = 'hotdog';

FoodCart

date	food	sold
02/25/08	pizza	349
02/26/08	hotdog	500
02/26/08	pizza	70

date	food	sold
02/25/08	pizza	349
02/26/08	pizza	70

Note: If the WHERE clause is omitted all rows of data are deleted from the table.

SQL Statements, Operations, Clauses

- SQL Statements:
  - Select
- SQL Operations:
  - Join
  - Left Join
  - Right Join
  - Like
- SQL Clauses:
  - Order By
  - Group By
  - Having

SQL: SELECT Statement

- A basic SELECT statement includes 3 clauses

SELECT <attribute name> FROM <tables> WHERE <condition>

SELECT	FROM	WHERE
Specifies the attributes that are part of the resulting relation	Specifies the tables that serve as the input to the statement	Specifies the selection condition, including the join condition.

Note: that you don't need to use WHERE

SQL: SELECT Statement (cont.)

- Using a "*" in a select statement indicates that every attribute of the input table is to be selected.

Example: SELECT * FROM … WHERE …;

- To get unique rows, type the keyword DISTINCT after SELECT.

Example: SELECT **DISTINCT** * FROM … WHERE …;

Example:

Person

Name	Age	Weight
Harry	34	80
Sally	28	64
George	29	70
Helena	54	54
Peter	34	80

1) SELECT *
FROM person
WHERE age > 30;

Name	Age	Weight
Harry	34	80
Helena	54	54
Peter	34	80

2) SELECT weight
FROM person
WHERE age > 30;

Weight
80
54
80

3) SELECT **distinct** weight
FROM person
WHERE age > 30;

Weight
80
54

SQL: Join operation

- A join can be specified in the FROM clause which list the two input relations and the WHERE clause which lists the join condition.

  Example:

Emp

ID	State
1000	CA
1001	MA
1002	TN

Dept

ID	Division
1001	IT
1002	Sales
1003	Biotech

SQL: Join operation (cont.)

- inner join = join

  SELECT *

  FROM emp join dept (or FROM emp, dept)

  on emp.id = dept.id;

Emp.ID	Emp.State	Dept.ID	Dept.Division
1001	MA	1001	IT
1002	TN	1002	Sales

SQL: Join operation (cont.)

- left outer join = left join

  SELECT *

  FROM emp left join dept

  on emp.id = dept.id;

Emp.ID	Emp.State	Dept.ID	Dept.Division
1000	CA	null	null
1001	MA	1001	IT
1002	TN	1002	Sales

SQL: Join operation (cont.)

- right outer join = right join

  SELECT *

  FROM emp right join dept

  on emp.id = dept.id;

Emp.ID	Emp.State	Dept.ID	Dept.Division
1001	MA	1001	IT
1002	TN	1002	Sales
null	null	1003	Biotech

SQL: Like operation

Pattern matching selection

- % (arbitrary string)

  SELECT *

  FROM emp

  WHERE ID like ‘%01’;

  ⇨ finds ID that ends with 01, e.g. 1001, 2001, etc

- _ (a single character)

  SELECT *

  FROM emp

  WHERE ID like ‘_01_’;

  ⇨ finds ID that has the second and third character as 01, e.g. 1010, 1011, 1012, 1013, etc

SQL: The ORDER BY Clause

Ordered result selection

- desc (descending order)

  SELECT *

  FROM emp

  order by state desc

  → puts state in descending order, e.g. TN, MA, CA

- asc (ascending order)

  SELECT *

  FROM emp

  order by id asc

  → puts ID in ascending order, e.g. 1001, 1002, 1003

SQL: The GROUP BY Clause

- The function to divide the tuples into groups and returns an aggregate for each group.
- Usually, it is an aggregate function's companion

```
SELECT food, sum(sold) as totalSold
FROM FoodCart
group by food;
```

FoodCart

date	food	sold
02/25/08	pizza	349
02/26/08	hotdog	500
02/26/08	pizza	70

food	totalSold
hotdog	500
pizza	419

SQL: The HAVING Clause

- The substitute of WHERE for aggregate functions
- Usually, it is an aggregate function's companion

```
SELECT food, sum(sold) as totalSold
FROM FoodCart
group by food
having sum(sold) > 450;
```

FoodCart

date	food	sold
02/25/08	pizza	349
02/26/08	hotdog	500
02/26/08	pizza	70

food	totalSold
hotdog	500

SQL: Aggregate Functions

Are used to provide summarization information for SQL statements, which return a single value.

- COUNT(attr)
- SUM(attr)
- MAX(attr)
- MIN(attr)
- AVG(attr)

Note: when using aggregate functions, NULL values are not considered, except in COUNT(*) .

SQL: Aggregate Functions (cont.)

FoodCart

date	food	sold
02/25/08	pizza	349
02/26/08	hotdog	500
02/26/08	pizza	70

- COUNT(attr) -> return # of rows that are not null

Ex: COUNT(distinct food) from FoodCart; -> 2

- SUM(attr) -> return the sum of values in the attr

Ex: SUM(sold) from FoodCart; -> 919

- MAX(attr) -> return the highest value from the attr

Ex: MAX(sold) from FoodCart; -> 500

SQL: Aggregate Functions (cont.)

FoodCart

date	food	sold
02/25/08	pizza	349
02/26/08	hotdog	500
02/26/08	pizza	70

- MIN(attr) -> return the lowest value from the attr

Ex: MIN(sold) from FoodCart; -> 70

- AVG(attr) -> return the average value from the attr

Ex: AVG(sold) from FoodCart; -> 306.33

Note: value is rounded to the precision of the datatype