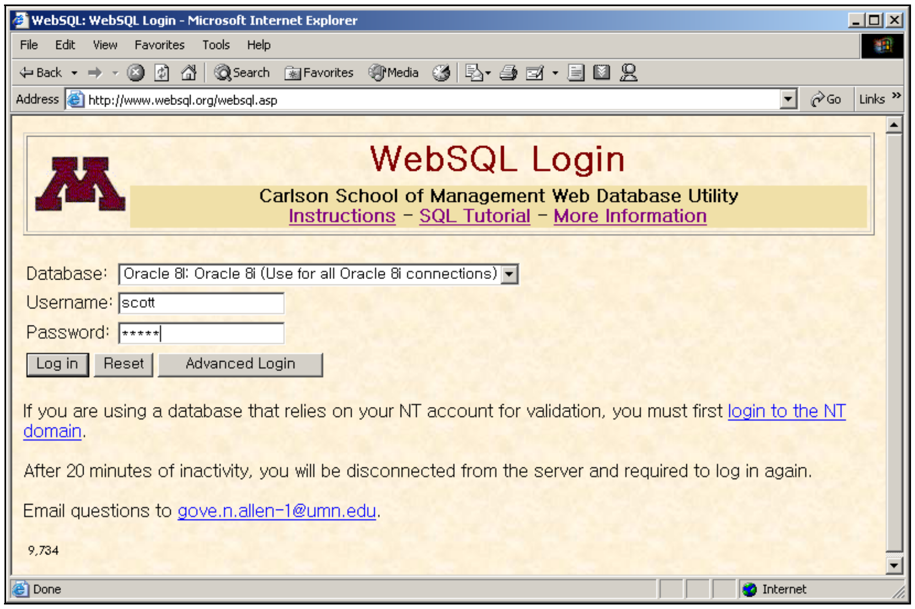
ORACLE SQL TUTORIAL USING WEBSQL

1. WEBSQL

WebSQL (http://www.websql.org) is an Internet front end for SQL compliant databases like Oracle. It was developed by Gove Allen, a recent Ph.D. graduate from the Carlson School of Management. Hosted by the University of Minnesota, it uses Microsoft's Active Server Pages (ASP) technology and Active Data Objects (ADO) to provide a SQL prompt to the database. It is intended as an instructional tool, but is also very handy for administering databases remotely.


WebSQL: WebSQL Login - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.websql.org/websql.asp
Go Links
Log in
Reset
Advanced Login
Done
Internet

WebSQL Login

Carlson School of Management Web Database Utility

Instructions – SQL Tutorial – More Information

Database: Oracle 8i: Oracle 8i (Use for all Oracle 8i connections)

Username: scott

Password: *****

If you are using a database that relies on your NT account for validation, you must first login to the NT domain.

After 20 minutes of inactivity, you will be disconnected from the server and required to log in again.

Email questions to gove.n.allen-1@umn.edu.

9,734

Figure 1: WebSQL Login Screenshot

1.1. INDIVIDUAL ACCOUNT

To create, delete, or modify your own tables (i.e., relational schema) on the database (e.g., Oracle), each group will have an account.

In addition, you may FTP your files (html, images, asp etc.) to your account and access your files on the web at http://www.internet-technology.org/i6201s03_accountname/path/filename.

2. SQL

SQL, which stands for *Structured Query Language*, is an ANSI (and ISO) standard language for querying, modifying, and managing relational databases. It is used to insert, delete, update, and retrieve data. **PL/SQL**, which stands for *Procedural Language/SQL*, is a proprietary language developed by Oracle as an extension to SQL. Like SQL, it also executes inside database. It has been created as a tool for coding business rules and procedures at the database level.

SQL can be divided into the following three languages: **DDL** (*Data Definition Language*), **DML** (*Data Manipulation Language*), and **DCL** (*Data Control Language*). DML (*Data Manipulation Language*) is the SQL statements that are used to select, insert, update, or delete the data from the table. DCL is the SQL sentences, which are used to control the behavior of these objects (e.g., commit, grant, rollback, etc).

In this tutorial, we will not cover PL/SQL due to space limitation. Rather, we will focus on the **DML** and **DDL**.

2.1. DDL

DDL (*Data Definition Language*) consists of a set of SQL statements that are used to create, alter, or drop the objects in the database. The objects include tables, indexes, clusters, sequences, triggers, procedures, functions, and packages. The **CREATE** command is used to create all of these objects. Likewise, the **ALTER** command is used to alter database objects, and the **DROP** command is used to delete them. Table 1 is a partial list of DDL statements.

SQL COMMAND	PURPOSE
ALTER PROCEDURE	Recompiles a stored procedure
ALTER TABLE	Adds a column, refines a column, changes storage allocation
ALTER TABLE ADD CONSTRAINT	Add a constraint to an existing table
ANALYZE	Gathers performance statistics for database objects
CREATE TABLE	Creates a table
CREATE INDEX	Creates an index
DROP INDEX	Drops an index
DROP TABLE	Drops a table from the database
GRANT	Grants privileges or roles to a user or another role
TRUNCATE	Deletes all the rows from a table
REVOKE	Removes privileges from a user or database role

Table 1: Partial List of DDL Statements

Alter, drop, and create are the most frequently used commands of the DDL. Therefore, I will focus on these three commands.

2.1.1. CREATE TABLE Statement

A table is the basic storage structure. A table (referred to as a **RELATION** in relational database terminology) consists of columns (or **attributes**) and contains rows (or **tuples**) of data. When creating a table, the user needs to specify each column by name and assign a data type for each column. There is no limit on the number of tables that can be created (of course, the operating system could impose restrictions based on available space).

The syntax for defining (or creating) a table is specified below. Note that the columns may be NULL columns or NOT NULL columns. If specified as a NULL column, the specified column can take on NULL values. Refer class notes for more information.

```
CREATE TABLE <[owner.]table_name>
(
        <column_name> <datatype> [DEFAULT expression] [column_constraint],
        <column_name> <datatype> [DEFAULT expression] [column_constraint], …,
        [table_constraint]
        [ENABLE <enable_clause>] [DISABLE <disable_clause>]
);
```

The column_constraint could be defined in many ways. The following example illustrates some of the ways in defining the column_constraint besides illustrating the CREATE TABLE command.

```
CREATE TABLE TITLES (
    title_id        CHAR (6)            NOT NULL,
    pub_id          CHAR (4)            NULL,
    title           VARCHAR2 (80)       NULL,
    title_type      CHAR (20)           NULL,
    price           NUMBER (6,2)        NULL,
    pubdate         DATE                NULL,
    CONSTRAINT titles_pk        PRIMARY KEY (title_id),
    CONSTRAINT titles_fk        FOREIGN KEY (pub_id) REFERENCES PUBLISHERS
);
```

We will discuss the primary key constraints and foreign key constraints in the following section. Note that the NOT NULL (and/or NULL) conditions can be specified as constraints. The advantage is that these can be enabled or disabled later if the need arises.

- *Data types (in Oracle)*

Some useful **data types** available in Oracle are listed in Table 2.

DATATYPE	DESCRIPTION
CHAR (n)	Fixed-length character data of length size bytes. Maximum size is 2000 bytes. Default and minimum size is 1 byte.
VARCHAR (n) **VARCHAR2 (n)**	Variable-length character string having maximum length size bytes. Maximum size is 4000, and minimum is 1. You must specify size for VARCHAR2. The VARCHAR datatype is currently synonymous with the VARCHAR2 datatype. Oracle recommends that you use VARCHAR2 rather than VARCHAR.
NUMBER (p, s)	Number having precision p and scale s. The precision p can range from 1 to 38. The scale s can range from -84 to 127.
NUMBER (n)	Integer values
NUMBER	Float, double precision values
DATE	For date and time values. Default format DD-MON-YY

Table 2: List of useful Oracle data types

- *Working with NUMBER Datatype – Scaling and Precision*

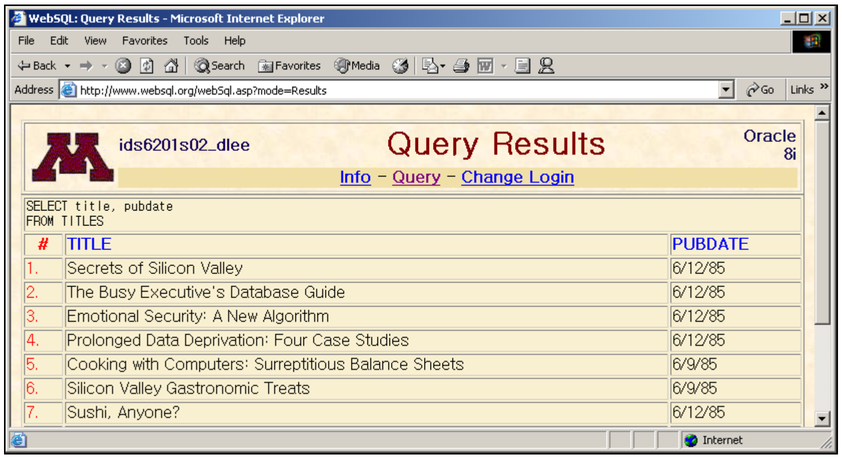
Specify the scale and precision of a fixed-point number column for extra integrity checking on input. Specifying scale and precision does not force all values to a fixed length. If a value exceeds the precision, Oracle returns an error. If a value exceeds the scale, Oracle rounds it.
Table 3 examples show how Oracle stores data using different precisions and scales.

RAW NUMBER	ORACLE DATATYPE	NUMBER STORED IN ORACLE
7456123.89	NUMBER	7456123.89
7456123.89	NUMBER (9)	7456124
7456123.89	NUMBER (9, 2)	7456123.89
7456123.89	NUMBER (9, 1)	7456123.9
7456123.89	NUMBER (7, -2)	7456100

Table 3: Scaling and Precision

- *Working with DATE Datatype*

The DATE datatype causes confusion for many users. Oracle's default format to display a date value as result of a **SELECT** statement is **DD/MM/YY** and the default format to accept a date value from **INSERT** statement is **DD-MON-YY**. Figure 2 illustrates an example of default DATE datatype generated from query.


WebSQL: Query Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.websql.org/webSql.asp?mode=Results
Go Links
ids6201s02_dlee
Query Results
Oracle 8i
Info - Query - Change Login
Internet

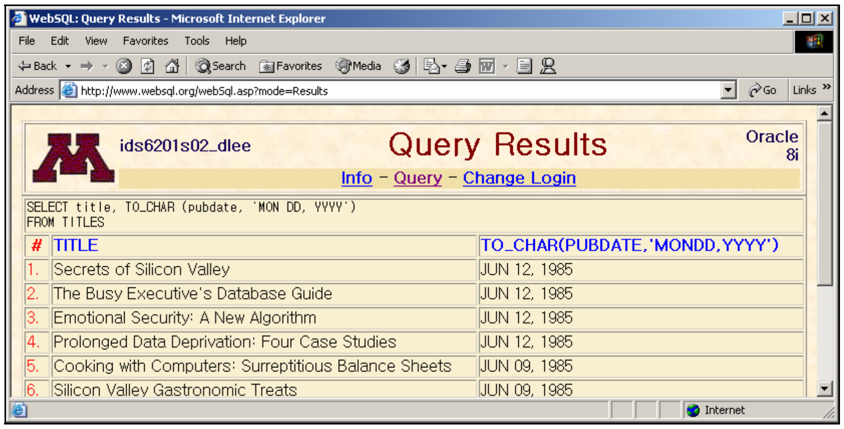
```
SELECT title, pubdate
FROM TITLES
```

#	TITLE	PUBDATE
1.	Secrets of Silicon Valley	6/12/85
2.	The Busy Executive's Database Guide	6/12/85
3.	Emotional Security: A New Algorithm	6/12/85
4.	Prolonged Data Deprivation: Four Case Studies	6/12/85
5.	Cooking with Computers: Surreptitious Balance Sheets	6/9/85
6.	Silicon Valley Gastronomic Treats	6/9/85
7.	Sushi, Anyone?	6/12/85

Figure 2: Query Results of Default DATE Datatype

If you want to see values in the *pubdate* column in the format of your choice, you have to convert these values into desired format by using the **TO_CHAR** function. The first parameter of the TO_CHAR function takes the value of the DATE datatype as input. The second parameter specifies the format in which the first parameter value has to be converted. The function converts the date value into desired format and returns the result as character datatype. Therefore, when you want see the *pubdate* of the TITLES in the format of DATE (MON DD, YYYY), you can change the format as illustrated in Figure 3.

```
SELECT title, TO_CHAR (pubdate, 'MON DD, YYYY')
FROM TITLES;
```


WebSQL: Query Results - Microsoft Internet Explorer
http://www.websql.org/webSql.asp?mode=Results
ids6201s02_dlee
Query Results
Oracle 8i
Info - Query - Change Login

```
SELECT title, TO_CHAR (pubdate, 'MON DD, YYYY')
FROM TITLES
```

#	TITLE	TO_CHAR(PUBDATE,'MONDD,YYYY')
1.	Secrets of Silicon Valley	JUN 12, 1985
2.	The Busy Executive's Database Guide	JUN 12, 1985
3.	Emotional Security: A New Algorithm	JUN 12, 1985
4.	Prolonged Data Deprivation: Four Case Studies	JUN 12, 1985
5.	Cooking with Computers: Surreptitious Balance Sheets	JUN 09, 1985
6.	Silicon Valley Gastronomic Treats	JUN 09, 1985

Figure 3: Query Results of Desired Format of DATE Datatype

Similarly you can enter date values in the DATE datatype column in any format you want, but you have to use the **TO_DATE** function to tell ORACLE what format you are using. If you are using the default format, you don't have to use TO_DATE function and can enter directly in single quotation marks. For example, you can add a new row in the *TITLES* table by giving the *pubdate* value in Oracle's default format (**18-JUN-85**) as follows:

```
INSERT INTO TITLES VALUES ('3021', '0877', 'The Gourmet Microwave', 'mod_cook', 2.99, 15000.00,
22246, 'Traditional French gourmet recipes.', '18-JUN-85')
```

The first parameter of the TO_DATE function takes a character datatype as input. The second parameter specifies the format in which the date is given in the first parameter value. The TO_DATE function uses the second parameter to understand the values given in the first parameter and converts it into the DATE datatype, which is returned as output by the function. The following INSERT statement inserts the date in the *pubdate* column.

```
INSERT INTO TITLES VALUES ('3021', '0877', 'The Gourmet Microwave', 'mod_cook', 2.99, 15000.00,
22246, 'Traditional French gourmet recipes.', TO_DATE ('06/18/1985','mm/dd/yyyy'))
```

2.1.2. Declarative Integrity Constraints

Declarative Integrity Constraints place restrictions on the data in a table. They offer the advantage of defining integrity controls in one step during the table creation process. You may specify the CONSTRAINT clauses as part of the CREATE TABLE command. It is a good idea to name your constraints to help you better identify and manage constraints. For example, if you wish to disable the constraint at a later time, you can identify the constraint easily if it has a unique name. All specified constraints are ENABLED by default.

An integrity constraint can be declared either at the column level or at the table level. A column constraint references and restricts only the column that it defined on; a table constraint restricts one or more columns in the table. The syntax for defining column constraints is slightly different from the syntax for defining table constraints (check previous section on CREATE TABLE).

There are five types of declarative integrity constraints: **UNIQUE**, **PRIMARY KEY**, **NULL/NOT NULL**, **Referential Integrity** and **CHECK** constraints.

- *Defining UNIQUE Constraint*

UNIQUE value constrains indicated column(s) so that no two rows can have the same non-null value. This constraint also creates a unique index on that column that can be dropped (or deleted) only if the constraint definition is dropped from the table. To set up a column to contain unique values, simply replace the keywords "PRIMARY KEY" by "UNIQUE." (Refer to the PRIMARY KEY Constraint)

- *Defining PRIMARY KEY Constraint*

The **PRIMARY KEY** constraint can be defined as a column constraint. If the primary key requires more than one column (attribute) of the table (relation) [=> **composite primary key**], we need to define it as a table-level constraint. In either case, the constraint is specified as part of the CREATE TABLE command or as part of the ALTER TABLE command. By defining this constraint, the NOT NULL and UNIQUE constraints are automatically enforced. Thus any primary key column(s) cannot have NULL values and the values must be unique (**Entity Integrity**). When a column is specified as the PRIMARY KEY, an index is created on this column automatically. This index is a UNIQUE INDEX.

The syntax for the table level constraint definition is as follows:

```
CONSTRAINT <constraint_name> PRIMARY KEY <column_list>
e.g.,      CONSTRAINT titles_pk  PRIMARY KEY (title_id)
```

while at the column level it is defined as

```
<column_name>  <datatype>  PRIMARY KEY
e.g.,      title_id   CHAR (6) PRIMARY KEY
```

- *Defining NULL / NOT NULL Constraints and DEFAULT Values*

The **NULL** / **NOT NULL** constraint is a column constraint. This constraint defines whether the column (attribute) can take on a NULL value (or NOT). Again, it is recommended that you specify this constraint as part of the CREATE TABLE command although it can be defined as part of the ALTER TABLE command as well.

```
CREATE TABLE [owner.]<tablename>
( <column_name> <datatype> [DEFAULT expression] [NOT NULL | NULL] [<column
constraint>]  );
```

By **default** columns are set up as "**NULL**", i.e. that the column *can* contain NULL values (except the primary key of course).
The [DEFAULT expression] clause specifies the value that the column (attribute) would take on if no value is specified at the time of data entry. The default value is specified as an expression (character strings, computed values etc.). If this is not specified, then the attribute value will be NULL (presuming NULL is acceptable).

```
pub_id   CHAR (4)              NULL,
title    VARCHAR2 (80)         DEFAULT "unknown" NOT NULL
```

- *Defining Referential Integrity Constraints on Columns*

The **referential integrity** constraint can also be defined as a column constraint. In such a case, it is associated with a particular column (attribute) of the table (relation). If the "**foreign key**" consists of more than one column, we need to define it as a **table-level constraint** (similar condition as with the primary key constraint). When you define a referential integrity constraint on a table, the current column(s) will refers to another set of columns (attributes) in some other table (relation), called the parent. The parent must be created and must exist when the referential constraint is defined.

```
CREATE TABLE [owner.]<tablename> ( <column_name> <datatype> [DEFAULT expression]
CONSTRAINT <constraint_name> REFERENCES <parent_table_name> [ON DELETE CASCADE |
ON DELETE SET NULL]);
```

By default Oracle assumes that the PRIMARY KEY column of the parent table is being referred to. The datatypes of columns in the parent and the child tables must be identical. Only columns that are specified as "PRIMARY KEY" or "UNIQUE" in the parent table may be referenced. Also, by default Oracle permits changes to the parent table, as long as it does not leave any rows in the child table without a referenced parent key value. The option "**ON DELETE CASCADE**" will delete all child tuples corresponding to a parent tuple being deleted. "**ON DELETE SET NULL**" will set the child the foreign key values in the child tuples to NULL upon deletion of a parent tuple (as long as the column allows NULL values).

```
CREATE TABLE SALESDETAILS (
    sonum               NUMBER NOT NULL,
    title_id            CHAR(6),
    store_id            CHAR (4),
    qty_ordered         NUMBER,
    qty_shipped         NUMBER,
    date_shipped        DATE,
    CONSTRAINT s_details_pk PRIMARY KEY (sonum, title_id, store_id),
    CONSTRAINT s_details_fk1 FOREIGN KEY (sonum, store_id) REFERENCES SALES ON
DELETE SET NULL,
    CONSTRAINT s_details_fk2 FOREIGN KEY (title_id) REFERENCES TITLES ON DELETE
CASCADE
);
```

- *Defining CHECK Constraints on Columns*

The **CHECK** constraint is used to restrict values that column (attribute) can take on. The restrictions can include: a list of constant expressions introduced with IN a set of conditions introduced with LIKE which may contain wildcard characters relational expressions. The CHECK constraint can be specified as part of the CREATE or ALTER TABLE commands only.

The following definition for a ‘TITLES’ table illustrates different types of constraints:

```
CREATE TABLE TITLES (
    title_id      CHAR (6)        CONSTRAINT titles_pk PRIMARY KEY,
    pub_id        CHAR (4),
    title VARCHAR2 (80)           DEFAULT "unknown" NOT NULL,
    title_type    CHAR (20),
    CONSTRAINT titles_fk FOREIGN KEY (pub_id) REFERENCES PUBLISHERS,
    CONSTRAINT title_type_constraint CHECK titile_type IN ('b', 'j', 'm')
);
```

Viewing Constraint definitions: All constraints are available in the user table USER_CONSTRAINTS. Any user can view constraints defined by him/her using the following SQL statement.

```
SELECT * FROM user_constraints;
```

2.1.3. ALTER TABLE Statement

The **ALTER TABLE** command is used to modify definitions of existing tables. You can use this command to

- add a new column
- add an integrity constraint
- redefine an existing column (datatype, size, default value)
- enable, disable, drop constraints and triggers.

Note: You cannot rename an existing column in the table with this command.

The syntax is:

```
ALTER TABLE <[owner.]tablename>
( [ADD (<column_name><datatype>[DEFAULT expr][<column constraint(s)>] ) ]
[MODIFY (<column_name><datatype>[DEFAULT expr][<column constraint(s)>] ) ]
[DROP <drop clause>]
[ENABLE <enable clause>]
[DISABLE <disable clause>]
);
```

At least one of (ADD, MODIFY, DROP, ENABLE, DISABLE) options must be specified.

- *Alters the TITLES table to **add a new column** called contract.*
  ALTER TABLE TITLES **ADD** (contract CHAR (1) NULL);
- **Changes the size of the column** "price" in the TITLES table.
  ALTER TABLE TITLES **MODIFY** (price DECIMAL (8,3));
- ***Disables*** the **constraint** defined in the previous example.
  ALTER TABLE TITLES **DISABLE CONSTRAINT** title_type_constraint;
- Drops the **constraint** defined in the previous example.
  ALTER TABLE TITLES **DROP CONSTRAINT** title_type_constraint;
- **Drops the column** price from the TITLES table
  ALTER TABLE TITLES **DROP** (price);

Note: Versions prior to Oracle 8*i* did not allow you to drop a column.

2.1.4. DROP Statement

The DROP command can be used to remove any existing object from the database. The command can be used to drop tables, views, indices, procedures, and triggers. Note that the execution of the 'DROP TABLE' command results in the removal of the data stored in that table as well. You have to own the table in order to drop the table. No other user will be permitted to drop the table owned by a particular user. The syntax and examples for the DROP command are as follows:

```
DROP [TABLE | VIEW | INDEX | PROCEDURE | TRIGGER] [owner.]<table_name>;
DROP TABLE TITLES;
```

2.1.5. CREATE TABLE AS <subquery>

This command is a variation of the CREATE TABLE command. It selects data from an existing table (the subquery) and creates a new table for that data. The command utilizes the column definitions of the existing table specified in the subquery. It does not change any datatypes or sizes and populates the new table with the data existing in the original table.

- *Create a new table called MAGAZINE by using the data in the TITLES.*

```
CREATE TABLE MAGAZNE (title_id, pub_id, title, price)
AS
(SELECT title_id, pub_id, title, price FROM TITLES);
```

2.2. DML

DML (*Data Manipulation Language*) is the SQL statements that are used to select, insert, update, or delete the data from the table. Just as the name implies, DML allows you to work with the contents of your database. Table 4 is a partial list of DML statements.

SQL COMMANDS	PURPOSE
INSERT	Add rows of data to a table
DELETE	Delete rows of data from a table
UPDATE	Change rows of data from a table
SELECT	Retrieve rows of data from a table/view
COMMIT	Make changes permanent (write to disk) for the current transaction(s)
ROLLBACK	Undo all changes since the last commit

Table 4: Partial List of DML Statements

2.2.1. INSERT Statement

I will cover INSERT command first because the other commands do you little good if there's no data in the database to manipulate. The insert statement is the direct way of populating tables using SQL. It inserts ONE row (tuple) at a time. The syntax for the INSERT statement is:

```
INSERT INTO <tablename> [<column list>] VALUES (<list of values>)
```

The list of values should consist of one value for each column (attribute) in the table. *If not all columns are being populated, the "column list" specifies the names of the columns for which corresponding values are being provided.* The value list (and the column list if supplied) needs to be enclosed within parentheses ( ). If the datatype of a column is CHAR, VARCHAR or DATE, the value corresponding to that column must be within single quotes ' '. Numeric data (INTEGER, FLOAT) should not be enclosed within quotes.

```
INSERT INTO TITLES VALUES ('8888', '1389', 'Secrets of Silicon Valley', 'popular_comp',
20.00, 4095, 'Muckraking reporting on the world''s largest computer hardware and software
manufacturers.', to_date('06/12/1985','mm/dd/yyyy'));
```

You can insert NULL values into columns using the method described below.

```
INSERT INTO PUBLISHERS (pub_id, pub_name) VALUES ('0736', 'New Age Books');
INSERT INTO PUBLISHERS VALUES ('0736', 'New Age Books', NULL, NULL, NULL);
```

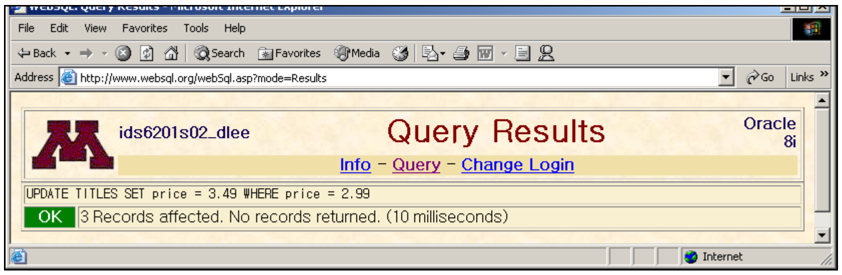
2.2.2. UPDATE Statement

This statement is used for modifying values in the relational tables. The syntax for the UPDATE statement is:

UPDATE <tablename> **SET** <column_name> = <value>, <column_name>= <value>
[**WHERE** <condition>]

The WHERE clause is optional. If the WHERE clause condition is specified only those rows that satisfy the given condition are updated. If no WHERE condition is specified, *all rows* in the table are updated. Hence you must be very careful when using the UPDATE statement.

UPDATE TITLES **SET** price = 3.49 **WHERE** price = 2.99;



```
UPDATE TITLES SET price = 3.49 WHERE price = 2.99
OK 3 Records affected. No records returned. (10 milliseconds)
```

Figure 4: Result of Update Statement

2.2.3. DELETE Statement

The DELETE statement is used to remove records from a table while keeping the table. The syntax for the DELETE statement is:

DELETE FROM <[owner.]tablename> [**WHERE** <condition>];

Like the UPDATE statement, if the WHERE condition is specified, the rows that meet the condition are removed. If the WHERE condition is not specified, all the rows in the table are removed. Be **very careful** using the DELETE statement. Note: if you accidentally delete rows you didn't want to, immediately run the ROLLBACK command.

DELETE FROM TITLEAUTHORS **WHERE** au_id='213-46-8915';
To undo the effects:
ROLLBACK;

Note: WebSQL does not support '**ROLLBACK**" function. When you execute any statements, it automatically **COMMIT** the commands.

2.2.4. SELECT Statement

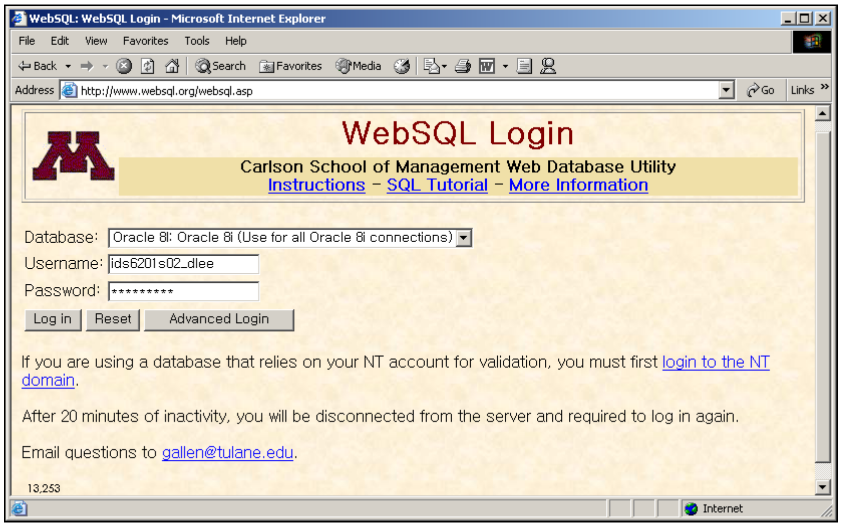
The purpose of the SELECT statement is to retrieve data from one or more tables or views. SELECT statement is the most frequently used statement because data is retrieved more often than inserted, deleted, or updated. The syntax for the SELECT statement is as follows:

```
SELECT [DISTINCT | ALL ] <list of columns to SELECT>
[FROM <List of table names>]
[WHERE <conditions>] [Operators used in specifying conditions include: AND, OR, NOT, =, < >,
>, <, >=, <=]
[GROUP BY <expression>]
[HAVING <condition>]
[ORDER BY [<column(s)> | <position>][ASC | DESC] ]
```

A SELECT statement can define restriction of rows through criteria based on an optional WHERE clause. The WHERE clause can contain comparisons such as =, >, <, >=, <=, !=, etc.

This section will use a set of tables containing some data on book publishing and sales collectively called the BOOKBIZ data. This data will be stored in Oracle database through WebSQL and will be used to help you understand and practice querying data using Oracle products. The relational description (DDL) of the BOOKBIZ data (the schema) is in APPENDIX 1 and a diagram of the tables is in APPENDIX 2.

To create BOOKBIZ database in your WebSQL account, go to www.websql.org with your username "**i6201s03_accountname**" and your password as shown in Figure 5.


WebSQL: WebSQL Login - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.websql.org/websql.asp
Go Links
Internet

WebSQL Login

Carlson School of Management Web Database Utility

Instructions - SQL Tutorial - More Information

Database: Oracle 8I: Oracle 8i (Use for all Oracle 8i connections)

Username: ids6201s02_dlee

Password: *********

Log in | Reset | Advanced Login

If you are using a database that relies on your NT account for validation, you must first login to the NT domain.

After 20 minutes of inactivity, you will be disconnected from the server and required to log in again.

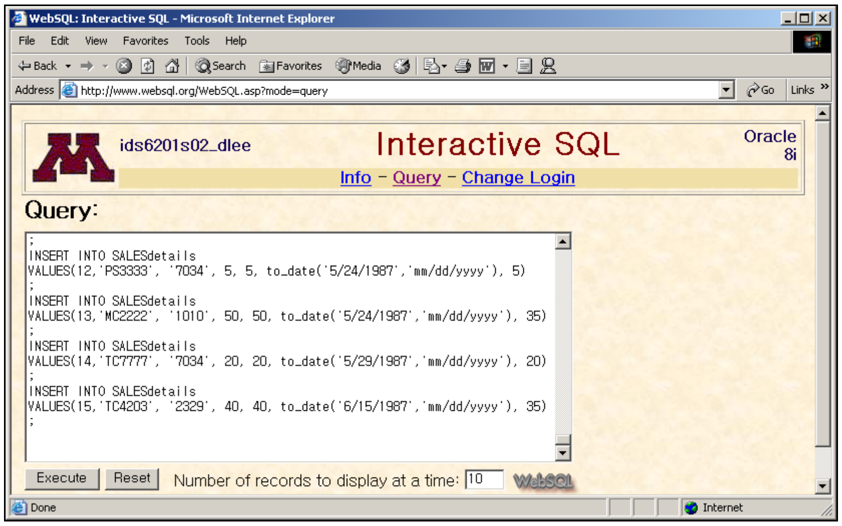
Email questions to gallen@tulane.edu.

13,253

Figure 5: WebSQL Login

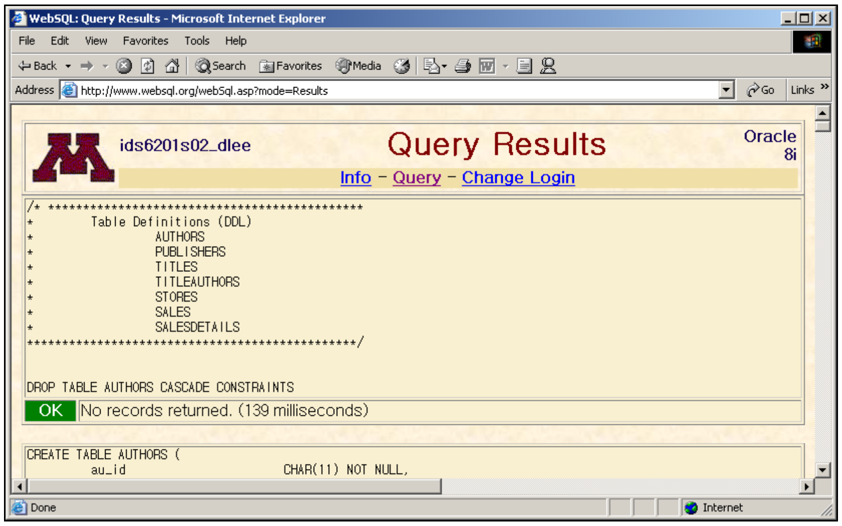
You need to download a file to contain SQL scripts for creating tables and populating data from the following url: http://mis4ever.com/classes/Bookbiz-oracle.txt.

Next, copy and paste the scripts from "Bookbiz-oracle.txt" to the textbox as shown in Figure 6, and execute the scripts. Then, all the tables and data will be created in the WebSQL as illustrated in Figure 7.


WebSQL: Interactive SQL - Microsoft Internet Explorer
http://www.websql.org/WebSQL.asp?mode=query
ids6201s02_dlee
Interactive SQL
Oracle 8i
Info - Query - Change Login
Query:
Execute
Reset
Number of records to display at a time: 10
Done
Internet

```
;
INSERT INTO SALESdetails
VALUES(12,'PS3333', '7034', 5, 5, to_date('5/24/1987','mm/dd/yyyy'), 5)
;
INSERT INTO SALESdetails
VALUES(13,'MC2222', '1010', 50, 50, to_date('5/24/1987','mm/dd/yyyy'), 35)
;
INSERT INTO SALESdetails
VALUES(14,'TC7777', '7034', 20, 20, to_date('5/29/1987','mm/dd/yyyy'), 20)
;
INSERT INTO SALESdetails
VALUES(15,'TC4203', '2329', 40, 40, to_date('6/15/1987','mm/dd/yyyy'), 35)
;
```

Figure 6: SQL Scripts


WebSQL: Query Results - Microsoft Internet Explorer
http://www.websql.org/webSql.asp?mode=Results
ids6201s02_dlee
Query Results
Oracle 8i
Info - Query - Change Login
Done
Internet

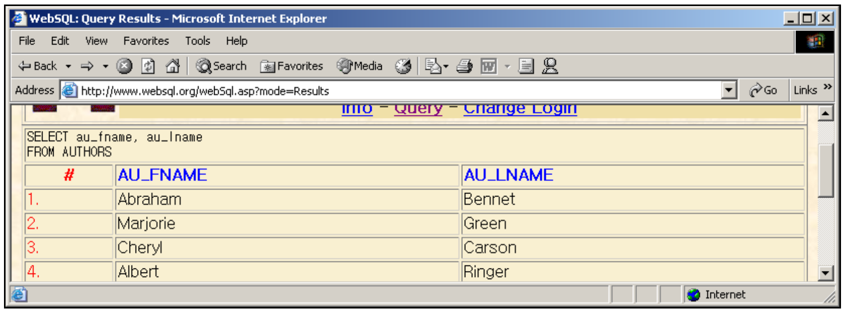
```
/* ***********************************************
*       Table Definitions (DDL)
*               AUTHORS
*               PUBLISHERS
*               TITLES
*               TITLEAUTHORS
*               STORES
*               SALES
*               SALESDETAILS
************************************************/

DROP TABLE AUTHORS CASCADE CONSTRAINTS
OK  No records returned. (139 milliseconds)

CREATE TABLE AUTHORS (
        au_id                   CHAR(11) NOT NULL,
```

Figure 7: Query Results

2.2.4.1. Single Table Queries

Q1. What are the first and last names of authors?

```
SELECT au_fname, au_lname
FROM AUTHORS;
```

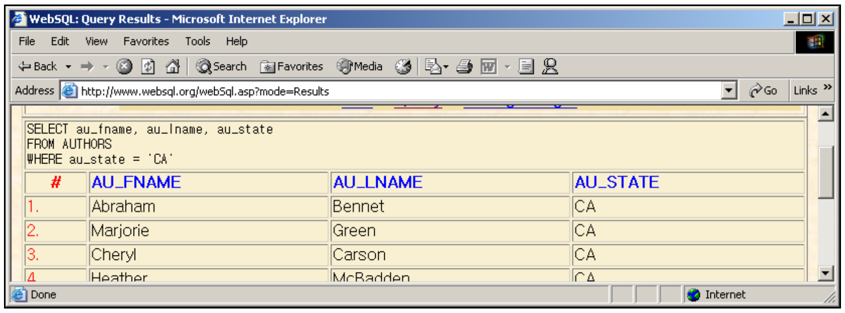


```
SELECT au_fname, au_lname
FROM AUTHORS
```

#	AU_FNAME	AU_LNAME
1.	Abraham	Bennet
2.	Marjorie	Green
3.	Cheryl	Carson
4.	Albert	Ringer

Q2. What are the names and states of authors living in California?

```
SELECT au_fname, au_lname, au_state
FROM AUTHORS
WHERE au_state = 'CA';
```

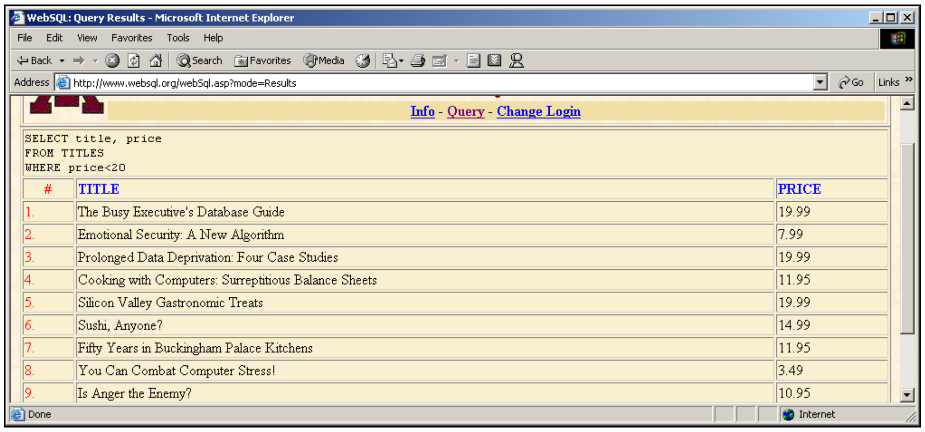


```
SELECT au_fname, au_lname, au_state
FROM AUTHORS
WHERE au_state = 'CA'
```

#	AU_FNAME	AU_LNAME	AU_STATE
1.	Abraham	Bennet	CA
2.	Marjorie	Green	CA
3.	Cheryl	Carson	CA
4.	Heather	McBadden	CA

Q3. What books cost less than $20?

```
SELECT title, price
FROM TITLES
WHERE price < 20;
```

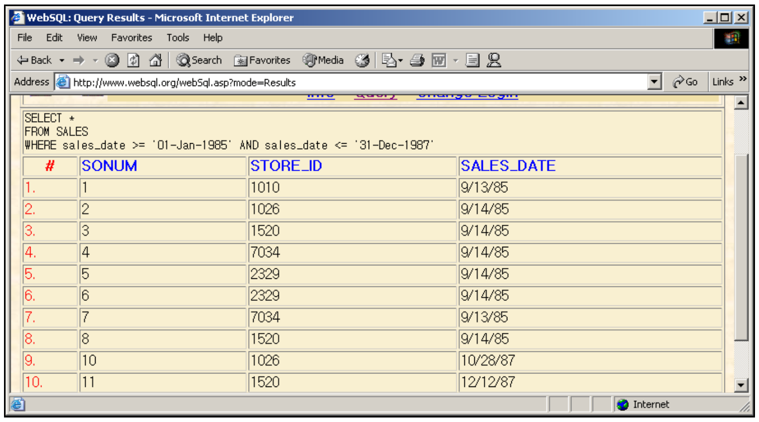

WebSQL: Query Results - Microsoft Internet Explorer
http://www.websql.org/webSql.asp?mode=Results
Info - Query - Change Login
Done
Internet

```
SELECT title, price
FROM TITLES
WHERE price<20
```

#	TITLE	PRICE
1.	The Busy Executive's Database Guide	19.99
2.	Emotional Security: A New Algorithm	7.99
3.	Prolonged Data Deprivation: Four Case Studies	19.99
4.	Cooking with Computers: Surreptitious Balance Sheets	11.95
5.	Silicon Valley Gastronomic Treats	19.99
6.	Sushi, Anyone?	14.99
7.	Fifty Years in Buckingham Palace Kitchens	11.95
8.	You Can Combat Computer Stress!	3.49
9.	Is Anger the Enemy?	10.95

Q4. What were the sales made between Jan-01-1985 and Dec-31-1987?

```
SELECT *
FROM SALES
WHERE sales_date >= '01-Jan-1985' AND sales_date <= '31-Dec-1987 ';
```


WebSQL: Query Results - Microsoft Internet Explorer
http://www.websql.org/webSql.asp?mode=Results
Internet

```
SELECT *
FROM SALES
WHERE sales_date >= '01-Jan-1985' AND sales_date <= '31-Dec-1987'
```

#	SONUM	STORE_ID	SALES_DATE
1.	1	1010	9/13/85
2.	2	1026	9/14/85
3.	3	1520	9/14/85
4.	4	7034	9/14/85
5.	5	2329	9/14/85
6.	6	2329	9/14/85
7.	7	7034	9/13/85
8.	8	1520	9/14/85
9.	10	1026	10/28/87
10.	11	1520	12/12/87

As an alternate usage, you can use the **BETWEEN** operator to get the same result as follows:

```
SELECT *
FROM SALES
WHERE sales_date BETWEEN '01-Jan-1985' AND '31-Dec-1987'
ORDER BY sales  date DESC:
```

- ***ORDER BY Clause***

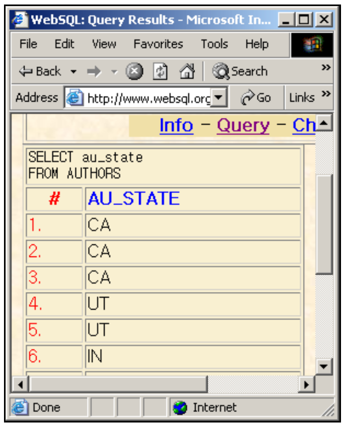
  OORDER BY must be the last clause of the SELECT statement. The name of the column that you want to use to order should follow the ORDER BY key word.

ORDER BY sales date / **ORDER BY** sales date **ASC** / ORDER BY sales date **DESC**

You can give more than one column name in an ORDER BY clause separated with commas. By default, ORDER BY statement retrieve rows in ascending order. You can also use ORDER BY for retrieving rows in descending order by using the **DESC** keyword.

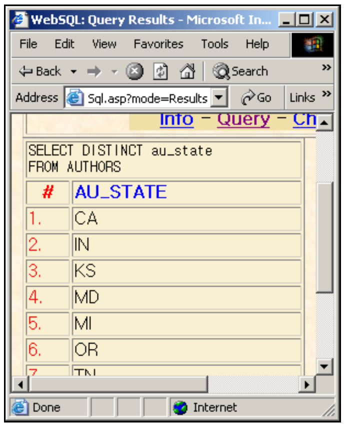
- ***Distinct Keyword***

  Using DISTINCT in SELECT statement prevents the same values to appear more than once.



```
SELECT au_state
FROM AUTHORS
```

#	AU_STATE
1.	CA
2.	CA
3.	CA
4.	UT
5.	UT
6.	IN



```
SELECT DISTINCT au_state
FROM AUTHORS
```

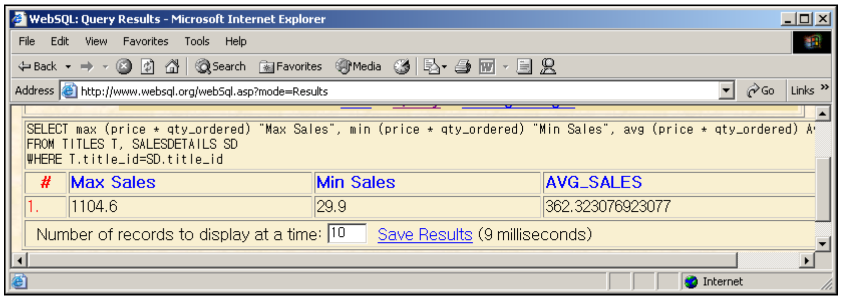
#	AU_STATE
1.	CA
2.	IN
3.	KS
4.	MD
5.	MI
6.	OR

2.2.4.2. Aliases

You can assign an **alias** to a **column** by writing a suitable alias after the column name in the SELECT list. If the alias contains more than one word, it should be enclosed in double quotation marks. You can also assign an **alias** to a **table name** and use the alias to specify which column belongs to which table.

Q5. What is the maximum quantity sold, the minimum quantity sold, and the average quantity ordered of all books. This also illustrates the use of Column Aliases. [Please observe what happens to the formatting of the column title in the output as well; is the formatting in effect for the rest of the session or just this query?]

SELECT max (price * qty_ordered) **"Max Sales",** min (price * qty_ordered) **"Min Sales"**, avg (price * qty_ordered) **Avg_Sales**
FROM TITLES T, SALESDETAILS SD
WHERE T.title_id=SD.title_id;



```
SELECT max (price * qty_ordered) "Max Sales", min (price * qty_ordered) "Min Sales", avg (price * qty_ordered) A
FROM TITLES T, SALESDETAILS SD
WHERE T.title_id=SD.title_id
```

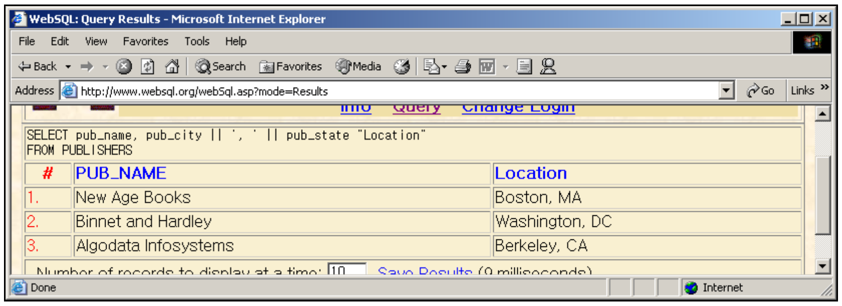
#	Max Sales	Min Sales	AVG_SALES
1.	1104.6	29.9	362.323076923077

Number of records to display at a time: 10 Save Results (9 milliseconds)

2.2.4.3. Concatenation

Q6. Demonstrate the use of Concatenation (also another example for aliases). Display for each publisher, the publisher name and location. The location should consist of the city and state concatenated in the form "City, State". Format the publisher name to a width of 40 characters, and the location to a width of 25.

```
SELECT pub_name, pub_city || ', ' || pub_state "Location"
FROM PUBLISHERS;
```



```
SELECT pub_name, pub_city || ', ' || pub_state "Location"
FROM PUBLISHERS
```

#	PUB_NAME	Location
1.	New Age Books	Boston, MA
2.	Binnet and Hardley	Washington, DC
3.	Algodata Infosystems	Berkeley, CA

2.2.4.4. Nested Queries (Subqueries)

Basically, instead of a single value to compare with in the WHERE clause, we want a set of values to compare with. Or we want to compare with the "results" of another query. **Subqueries** (i.e., **nested queries**) are important in the sense that they can handle very complex tasks. So we somehow specify a set (either directly or by running a query). Then we "compare" values from that set in our 'WHERE' clause. Most tasks that can be performed by using JOIN operations can also be performed by using subqueries.

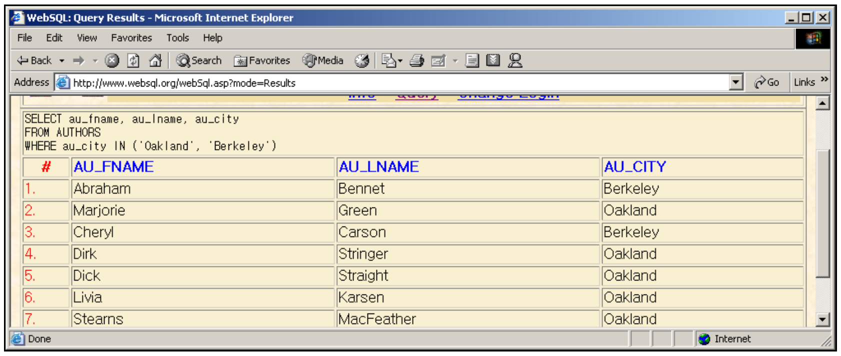
- ***IN / NOT IN Operator***

  If for any row, the value of the given column matches any of the values given in parentheses after the **IN** key word, the row is displayed. Similarly, you can use **NOT IN** operator if you want to see the information except for given values in parentheses.

To illustrate the concept of the "**inner set values**"

Q7. What are the names of authors living in Oakland or Berkeley?

```
SELECT au_fname, au_lname, au_city
FROM AUTHORS
WHERE au_city IN ('Oakland ', 'Berkeley ');
```



```
SELECT au_fname, au_lname, au_city
FROM AUTHORS
WHERE au_city IN ('Oakland', 'Berkeley')
```

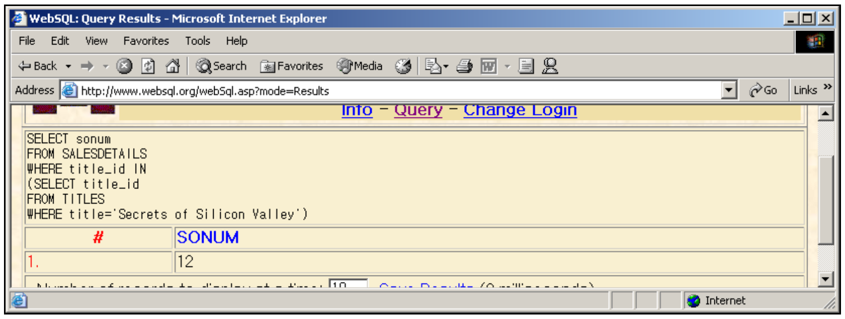
#	AU_FNAME	AU_LNAME	AU_CITY
1.	Abraham	Bennet	Berkeley
2.	Marjorie	Green	Oakland
3.	Cheryl	Carson	Berkeley
4.	Dirk	Stringer	Oakland
5.	Dick	Straight	Oakland
6.	Livia	Karsen	Oakland
7.	Stearns	MacFeather	Oakland

*So, if we wanted the "**inner set**" to contain the title_id of the book 'Secrets of Silicon Valley', we'd have it as:*

```
SELECT title_id
FROM TITLES
WHERE title='Secrets of Silicon Valley';
```

Q8. Use above to Find sales numbers that have sold the book:

```
SELECT sonum
FROM SALESDETAILS
WHERE title_id IN
(SELECT title_id
FROM TITLES
WHERE title='Secrets of Silicon Valley');
```


WebSQL: Query Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.websql.org/webSql.asp?mode=Results
Go Links
Info - Query - Change Login
Internet

```
SELECT sonum
FROM SALESDETAILS
WHERE title_id IN
(SELECT title_id
FROM TITLES
WHERE title='Secrets of Silicon Valley')
```

#	SONUM
1.	12

2.2.4.5. Using JOIN Operations

A JOIN operation is a mechanism of relating two or more tables by giving the joining conditions. JOIN operations can be divided into three major types: **Equi-Joins**, **Self-Joins**, and **Outer-Joins**.

- ***EUQI-JOINS***

  Equi-Join is a type of join using equality comparisons to join two or more tables. That is, the foreign key of one table is equated with the parent key of another table. To join tables, you include their names in the FROM clause separated with commas. Equi-Join is the most frequently used among the JOIN operations.

- ***SELF-JOINS***

  Self-Join is used when a table relates back to itself. This can happen in an EMPLOYEE table where the department and title information is included (for example, when a manager is also managed).

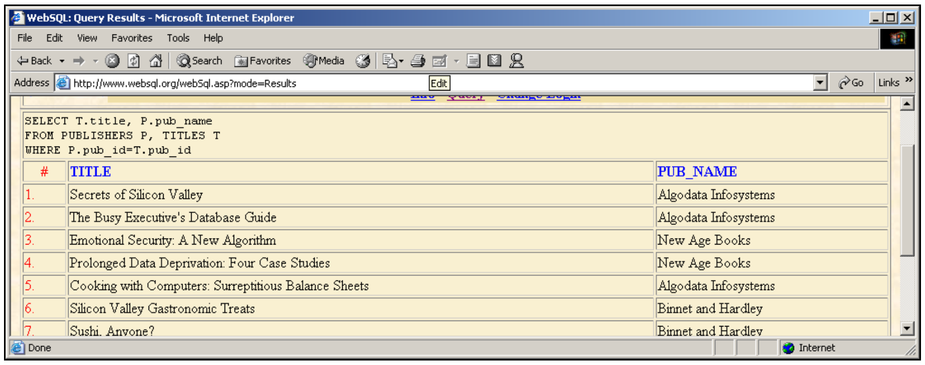
```
SELECT A. name Employee, B.name Manager
FROM EMPLOYEES A, EMPLOYEES B
WHERE A.super_id = B.id;
```

- ***OUTER-JOINS***

  Outer-Join is a type of join in which data not meeting the join criteria (i.e., the join vlaue is NULL) is also returned by the query. Suppose that you have a department that has no employees yet, and you want the department to be listed in a report. In this case, an Outer-Join would be required.An outer join is signified by using the plus sign inside parenthese **(+)** to indicate the outer join column for the table deficient in data.

Q9. Display the full name of the publisher for each title

```
SELECT T.title, P.pub_name
FROM PUBLISHERS P, TITLES T
WHERE P.pub_id = T.pub_id;
```

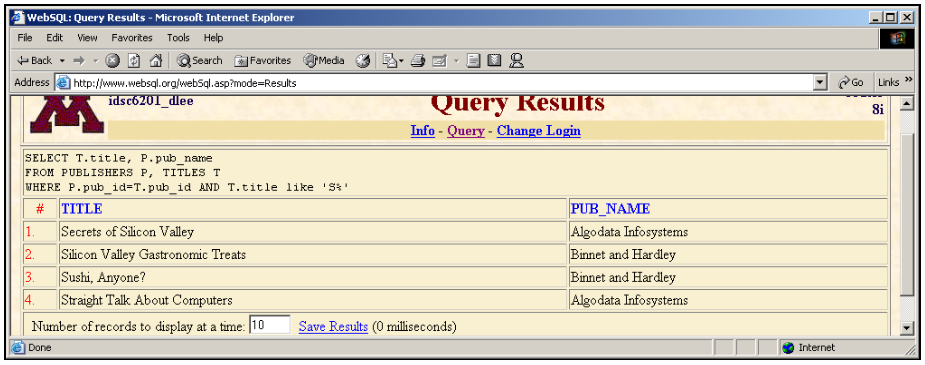


```
SELECT T.title, P.pub_name
FROM PUBLISHERS P, TITLES T
WHERE P.pub_id=T.pub_id
```

#	TITLE	PUB_NAME
1.	Secrets of Silicon Valley	Algodata Infosystems
2.	The Busy Executive's Database Guide	Algodata Infosystems
3.	Emotional Security: A New Algorithm	New Age Books
4.	Prolonged Data Deprivation: Four Case Studies	New Age Books
5.	Cooking with Computers: Surreptitious Balance Sheets	Algodata Infosystems
6.	Silicon Valley Gastronomic Treats	Binnet and Hardley
7.	Sushi, Anyone?	Binnet and Hardley

Q10. How about, just for titles beginning with the letter 'S'?

```
SELECT T.title, P.pub_name
FROM PUBLISHERS P, TITLES T
WHERE P.pub_id=T.pub_id AND T.title LIKE 'S%';
```



```
SELECT T.title, P.pub_name
FROM PUBLISHERS P, TITLES T
WHERE P.pub_id=T.pub_id AND T.title like 'S%'
```

#	TITLE	PUB_NAME
1.	Secrets of Silicon Valley	Algodata Infosystems
2.	Silicon Valley Gastronomic Treats	Binnet and Hardley
3.	Sushi, Anyone?	Binnet and Hardley
4.	Straight Talk About Computers	Algodata Infosystems

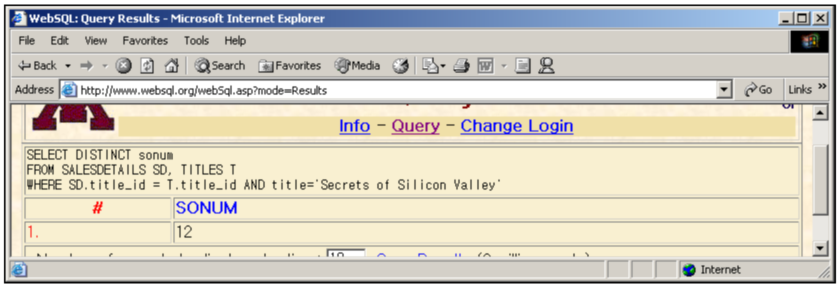
Number of records to display at a time: 10 Save Results (0 milliseconds)

▪ ***Like Operator***

You have to rely on the **LIKE** operator when you are not sure of the exact spelling of any word in the database. The **%** and _ (underscore) characters are available with the LIKE operator. Compared to %, _ designates only one letter.

*Q11. An example you could do using **nested queries or with a join**: Sales Numbers that have sold the book: 'Secrets of Silicon Valley'*

```
SELECT DISTINCT sonum
FROM SALESDETAILS SD, TITLES T
WHERE SD.title_id = T.title_id AND title='Secrets of Silicon Valley';
```

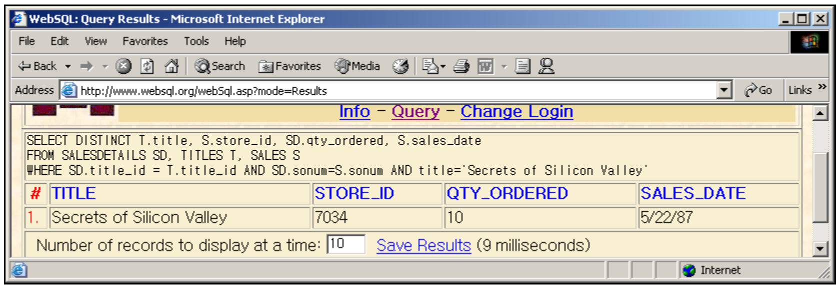


```
SELECT DISTINCT sonum
FROM SALESDETAILS SD, TITLES T
WHERE SD.title_id = T.title_id AND title='Secrets of Silicon Valley'
```

#	SONUM
1.	12

Q12. ***3 tables, 2 joins****: What about also showing the numbers of the Stores, titles, ordered quantity, and sales date?*

SELECT DISTINCT T.title, S.store_id, SD.qty_ordered, S.sales_date
FROM SALESDETAILS SD, TITLES T, SALES S
WHERE SD.title_id = T.title_id AND SD.sonum=S.sonum AND title='Secrets of Silicon Valley';



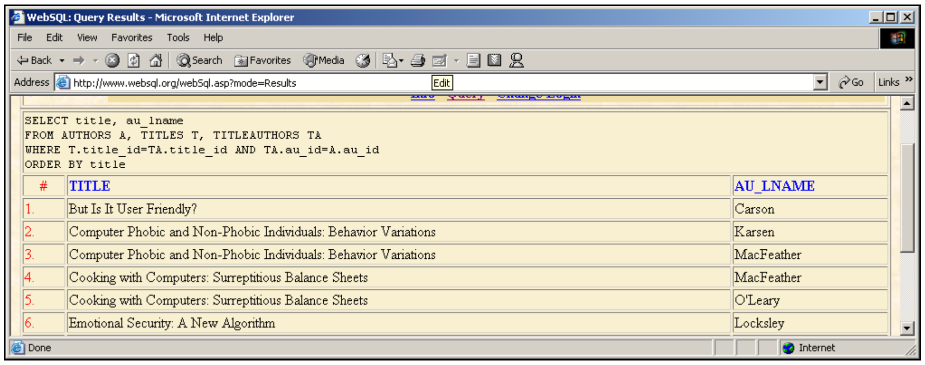
```
SELECT DISTINCT T.title, S.store_id, SD.qty_ordered, S.sales_date
FROM SALESDETAILS SD, TITLES T, SALES S
WHERE SD.title_id = T.title_id AND SD.sonum=S.sonum AND title='Secrets of Silicon Valley'
```

#	TITLE	STORE_ID	QTY_ORDERED	SALES_DATE
1.	Secrets of Silicon Valley	7034	10	5/22/87

Number of records to display at a time: 10 Save Results (9 milliseconds)

Q13. E.g. 2, with 3 tables: display for each book the last name of the author.

SELECT title, au_lname
FROM AUTHORS A, TITLES T, TITLEAUTHORS TA
WHERE T.title_id=TA.title_id AND TA.au_id=A.au_id
ORDER BY title;

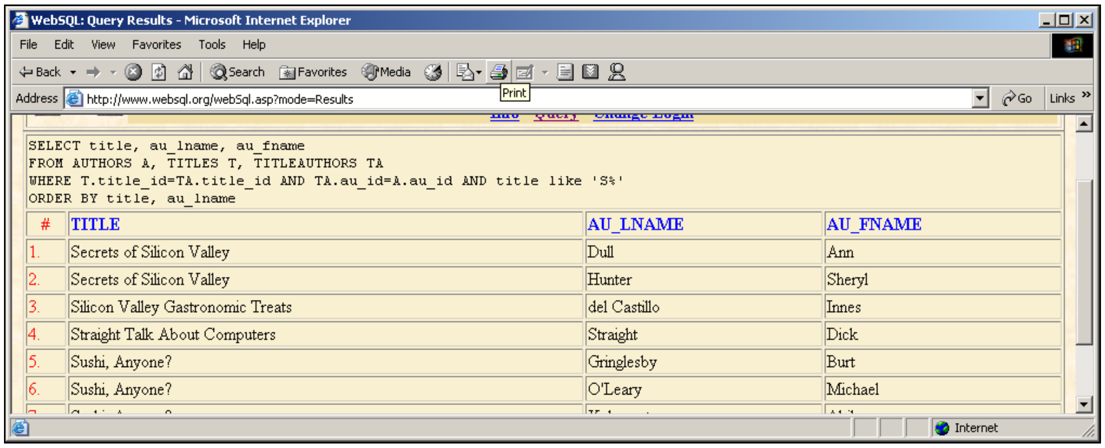


```
SELECT title, au_lname
FROM AUTHORS A, TITLES T, TITLEAUTHORS TA
WHERE T.title_id=TA.title_id AND TA.au_id=A.au_id
ORDER BY title
```

#	TITLE	AU_LNAME
1.	But Is It User Friendly?	Carson
2.	Computer Phobic and Non-Phobic Individuals: Behavior Variations	Karsen
3.	Computer Phobic and Non-Phobic Individuals: Behavior Variations	MacFeather
4.	Cooking with Computers: Surreptitious Balance Sheets	MacFeather
5.	Cooking with Computers: Surreptitious Balance Sheets	O'Leary
6.	Emotional Security: A New Algorithm	Locksley

Q14. Demonstrate the use of additional conditions in the WHERE (besides join). For each title beginning with the letter 'S', display the last and first names of the authors. Ensure the output is sorted alphabetically: first by title, then by author last name.

```
SELECT title, au_lname, au_fname
FROM AUTHORS A, TITLES T, TITLEAUTHORS TA
WHERE T.title_id=TA.title_id AND TA.au_id=A.au_id AND title like 'S%'
ORDER BY title, au_lname;
```

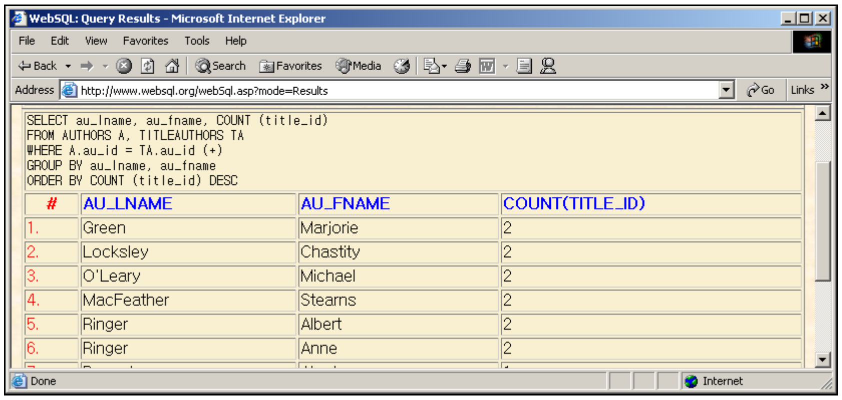


```
SELECT title, au_lname, au_fname
FROM AUTHORS A, TITLES T, TITLEAUTHORS TA
WHERE T.title_id=TA.title_id AND TA.au_id=A.au_id AND title like 'S%'
ORDER BY title, au_lname
```

#	TITLE	AU_LNAME	AU_FNAME
1.	Secrets of Silicon Valley	Dull	Ann
2.	Secrets of Silicon Valley	Hunter	Sheryl
3.	Silicon Valley Gastronomic Treats	del Castillo	Innes
4.	Straight Talk About Computers	Straight	Dick
5.	Sushi, Anyone?	Gringlesby	Burt
6.	Sushi, Anyone?	O'Leary	Michael

*Q15. **Outer-Join Example**: Suppose you want a list of all authors, including those whose books were not published yet.*

```
SELECT au_lname, au_fname, COUNT (title_id)
FROM AUTHORS A, TITLEAUTHORS TA
WHERE A.au_id = TA.au_id (+)
GROUP BY au_lname, au_fname
ORDER BY COUNT (title_id) DESC;
```



```
SELECT au_lname, au_fname, COUNT (title_id)
FROM AUTHORS A, TITLEAUTHORS TA
WHERE A.au_id = TA.au_id (+)
GROUP BY au_lname, au_fname
ORDER BY COUNT (title_id) DESC
```

#	AU_LNAME	AU_FNAME	COUNT(TITLE_ID)
1.	Green	Marjorie	2
2.	Locksley	Chastity	2
3.	O'Leary	Michael	2
4.	MacFeather	Stearns	2
5.	Ringer	Albert	2
6.	Ringer	Anne	2

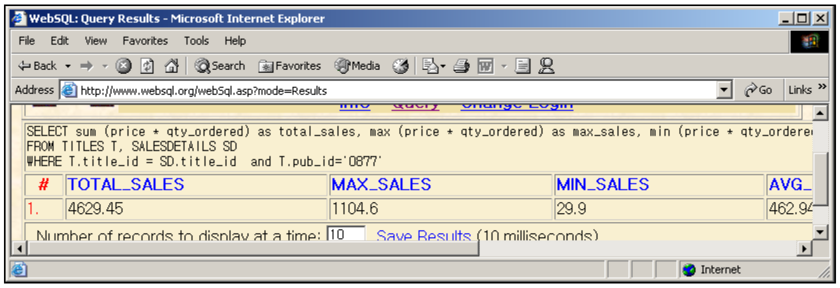
2.2.4.6. Group Function - Aggregate Operators

FUNCTION	RETURN VALUE
SUM (n)	Summed Value
AVG (n)	Average Value
COUNT (n)	Number of Rows
MAX (n)	Maximum Value
MIN (n)	Minimum Value

Table 5: Partial List of Group Functions

Q16. Let's get statistics for publisher '0877', such as total sales, max sales, min sales, average sales, etc.

```
SELECT sum (price * qty_ordered) as total_sales, max (price * qty_ordered) as max_sales, min
(price * qty_ordered) as min_sales, avg (price * qty_ordered) as avg_sales, count (SD.title_id)
FROM TITLES T, SALESDETAILS SD
WHERE T.title_id = SD.title_id  and T.pub_id='0877';
```


WebSQL: Query Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.websql.org/webSql.asp?mode=Results
Go Links
Internet

```
SELECT sum (price * qty_ordered) as total_sales, max (price * qty_ordered) as max_sales, min (price * qty_ordere
FROM TITLES T, SALESDETAILS SD
WHERE T.title_id = SD.title_id  and T.pub_id='0877'
```

#	TOTAL_SALES	MAX_SALES	MIN_SALES	AVG_
1.	4629.45	1104.6	29.9	462.94

Number of records to display at a time: 10 Save Results (10 milliseconds)

2.2.4.7. GROUP BY Clause

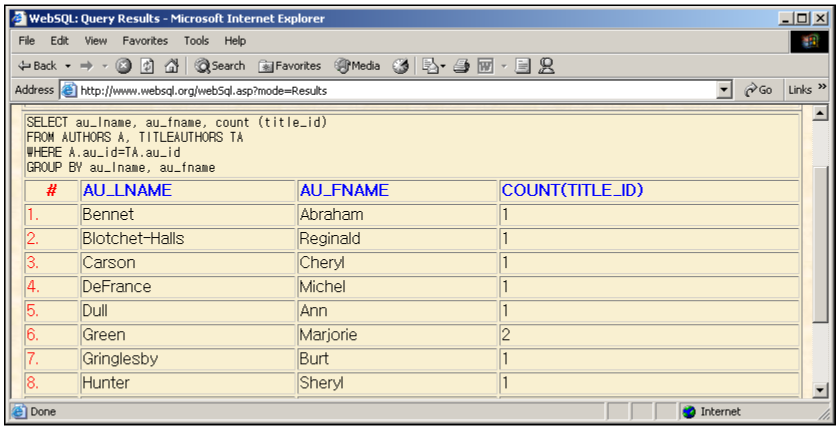
You can split a table into groups and use a group function on each group. Suppose that you want to find the sum of salaries of each department rather than the whole company's sum of salaries. Therefore, you should use the **Group BY** clause to split the EMPLYEES table into as many groups as there are distinct department numbers, and then apply the SUM group function to each group. The statement's GROUP BY clause tells Oracle that you want the sum of salaries for each department.

```
SELECT dept_no, SUM (sal)
FROM EMPLOYEES
GROUP BY dept_no;
```

You can also use the WHERE clause when using GROUP BY.

*Q17. Using **GROUP BY**: For each author display their name and the number of books published. Note the correspondence between the SELECT and GROUP BY clauses.*

```
SELECT au_lname, au_fname, count (title_id)
FROM AUTHORS A, TITLEAUTHORS TA
WHERE A.au_id=TA.au_id
GROUP BY au_lname, au_fname;
```


WebSQL: Query Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.websql.org/webSql.asp?mode=Results
Go Links
Done
Internet

```
SELECT au_lname, au_fname, count (title_id)
FROM AUTHORS A, TITLEAUTHORS TA
WHERE A.au_id=TA.au_id
GROUP BY au_lname, au_fname
```

#	AU_LNAME	AU_FNAME	COUNT(TITLE_ID)
1.	Bennet	Abraham	1
2.	Blotchet-Halls	Reginald	1
3.	Carson	Cheryl	1
4.	DeFrance	Michel	1
5.	Dull	Ann	1
6.	Green	Marjorie	2
7.	Gringlesby	Burt	1
8.	Hunter	Sheryl	1

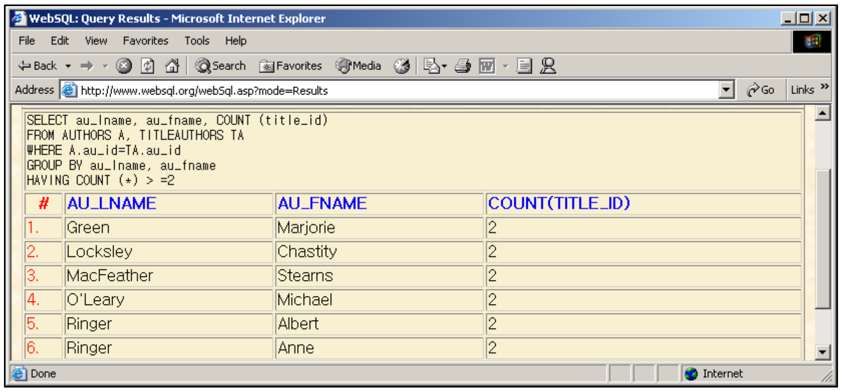
2.2.4.8. *HAVING Clause*

As you have seen, you can split a table into different groups and then apply group functions on individual groups. You can further apply group functions on individual groups. You can further restrict the group results returned by the group function by using the **HAVING** clause.

*Q18. Example Extension: Using **Group By & Having**: For authors who have published two or more books, display their name and the number of books published.*

```
SELECT au_lname, au_fname, COUNT (title_id)
FROM AUTHORS A, TITLEAUTHORS TA
WHERE A.au_id=TA.au_id
GROUP BY au_lname, au_fname
HAVING COUNT (*) > =2;
```

Note: You can't use a HAVING clause if the GROUP BY clause isn't used. If your criterion involves group functions, it must be placed in the HAVING clause, as in COUNT (*) >= 2. If your criterion involves a single-row function orcolumn names, it must be placed in the WHERE clause.

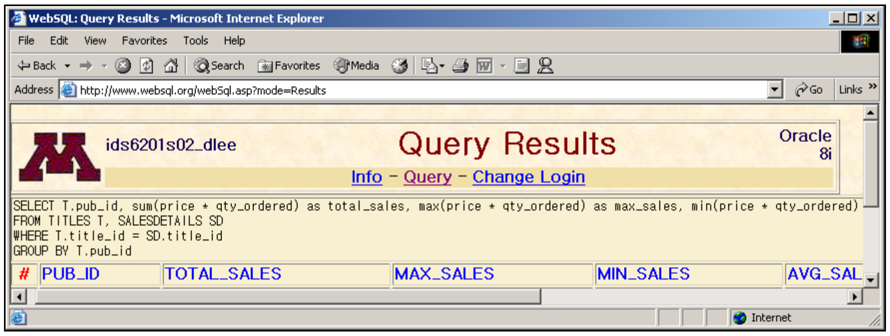

WebSQL: Query Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.websql.org/webSql.asp?mode=Results
Go Links
Done
Internet

```
SELECT au_lname, au_fname, COUNT (title_id)
FROM AUTHORS A, TITLEAUTHORS TA
WHERE A.au_id=TA.au_id
GROUP BY au_lname, au_fname
HAVING COUNT (*) > =2
```

#	AU_LNAME	AU_FNAME	COUNT(TITLE_ID)
1.	Green	Marjorie	2
2.	Locksley	Chastity	2
3.	MacFeather	Stearns	2
4.	O'Leary	Michael	2
5.	Ringer	Albert	2
6.	Ringer	Anne	2

Q19. Now, how about for the statistics of all publishers?

```
SELECT T.pub_id, sum(price * qty_ordered) as total_sales, max(price * qty_ordered) as
max_sales, min(price * qty_ordered) as min_sales, avg(price * qty_ordered) as avg_sales,
count(SD.title_id)
FROM TITLES T, SALESDETAILS SD
WHERE T.title_id = SD.title_id
GROUP BY T.pub_id
```

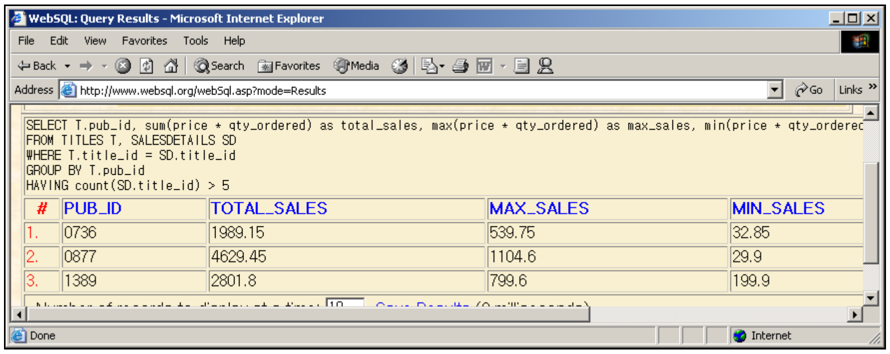

WebSQL: Query Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.websql.org/webSql.asp?mode=Results
Go Links
ids6201s02_dlee
Query Results
Oracle 8i
Info - Query - Change Login
Internet

```
SELECT T.pub_id, sum(price * qty_ordered) as total_sales, max(price * qty_ordered) as max_sales, min(price * qty_ordered)
FROM TITLES T, SALESDETAILS SD
WHERE T.title_id = SD.title_id
GROUP BY T.pub_id
```

#	PUB_ID	TOTAL_SALES	MAX_SALES	MIN_SALES	AVG_SAL

Q20. Now, how about for all publishers with more than 5 books?

```
SELECT T.pub_id, sum(price * qty_ordered) as total_sales, max(price * qty_ordered) as
max_sales, min(price * qty_ordered) as min_sales, avg(price * qty_ordered) as avg_sales
FROM TITLES T, SALESDETAILS SD
WHERE T.title_id = SD.title_id
GROUP BY T.pub_id
HAVING count(SD.title_id) > 5;
```


WebSQL: Query Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.websql.org/webSql.asp?mode=Results
Go Links
Done
Internet

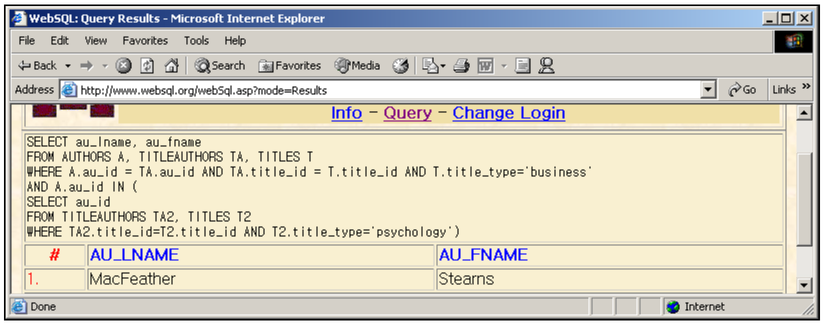
```
SELECT T.pub_id, sum(price * qty_ordered) as total_sales, max(price * qty_ordered) as max_sales, min(price * qty_ordered
FROM TITLES T, SALESDETAILS SD
WHERE T.title_id = SD.title_id
GROUP BY T.pub_id
HAVING count(SD.title_id) > 5
```

#	PUB_ID	TOTAL_SALES	MAX_SALES	MIN_SALES
1.	0736	1989.15	539.75	32.85
2.	0877	4629.45	1104.6	29.9
3.	1389	2801.8	799.6	199.9

2.2.4.9. Nested Queries (Subqueries) with JOINS

Q21. Suppose we wanted to list authors who had had written a book (or books) in the category of 'business', as well as in the category of 'psychology'.

```
SELECT au_lname, au_fname
FROM AUTHORS A, TITLEAUTHORS TA, TITLES T
WHERE A.au_id = TA.au_id AND TA.title_id = T.title_id AND T.title_type='business'
AND A.au_id IN (
SELECT au_id
FROM TITLEAUTHORS TA2, TITLES T2
WHERE TA2.title_id=T2.title_id AND T2.title_type='psychology');
```


WebSQL: Query Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.websql.org/webSql.asp?mode=Results
Go Links
Info – Query – Change Login
Done
Internet

```
SELECT au_lname, au_fname
FROM AUTHORS A, TITLEAUTHORS TA, TITLES T
WHERE A.au_id = TA.au_id AND TA.title_id = T.title_id AND T.title_type='business'
AND A.au_id IN (
SELECT au_id
FROM TITLEAUTHORS TA2, TITLES T2
WHERE TA2.title_id=T2.title_id AND T2.title_type='psychology')
```

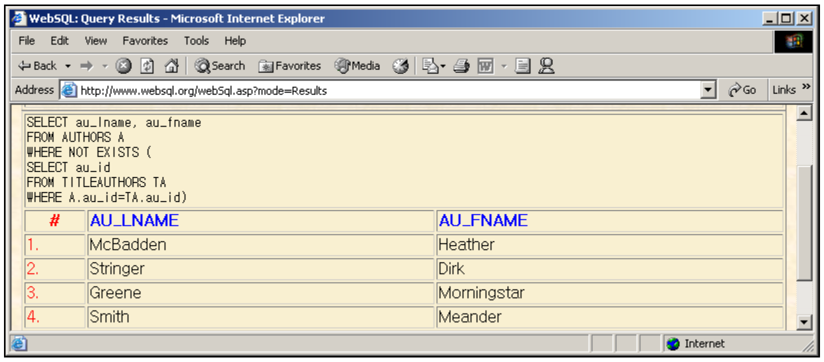
#	AU_LNAME	AU_FNAME
1.	MacFeather	Stearns

2.2.4.10. EXISTS / NOT EXISTS Operators

The **EXISTS** operator is useful when you are not concerned with what value is returned by a subquery but are concerned only with whether the subquery returns any row. If the subquery has returned at least one row, the EXISTS operator returns true to the main query's WHERE clause and the row is included in the result. On the other hand, **NOT EXISTS** returns true if no rows are returned by the subquery and returns false when at least one row is returned by the subquery.

Q22. *Once again, suppose we wanted to list authors who hadn't published a book yet. Or to look at it another way: authors in AUTHORS who did not have a corresponding entry in TITLEAUTHORS, i.e. there does NOT EXIST an entry in TITLEAUTHORS for that AUTHORS. (Note: the connection / join between the author in the inner & outer queries.)*

```
SELECT au_lname, au_fname
FROM AUTHORS A
WHERE NOT EXISTS (
SELECT au_id
FROM TITLEAUTHORS TA
WHERE A.au_id=TA.au_id);
```

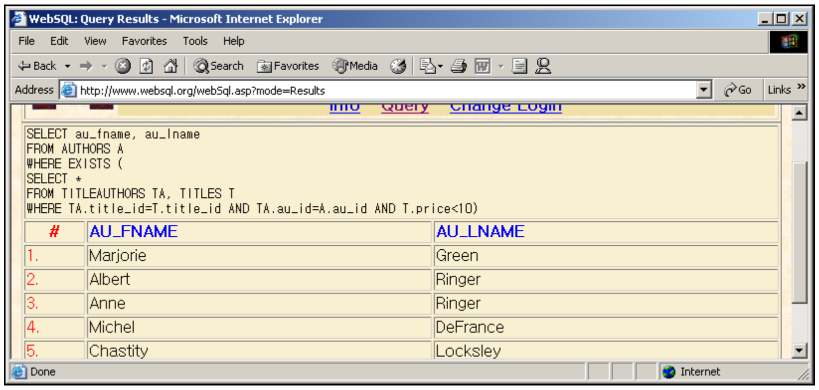

WebSQL: Query Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.websql.org/webSql.asp?mode=Results
Go Links
Internet

```
SELECT au_lname, au_fname
FROM AUTHORS A
WHERE NOT EXISTS (
SELECT au_id
FROM TITLEAUTHORS TA
WHERE A.au_id=TA.au_id)
```

#	AU_LNAME	AU_FNAME
1.	McBadden	Heather
2.	Stringer	Dirk
3.	Greene	Morningstar
4.	Smith	Meander

Q23. An example for EXISTS: List names of authors (first & last) who have written books priced less than $10; i.e. there exists a book that this author has written and is priced less than $10.

```
SELECT au_fname, au_lname
FROM AUTHORS A
WHERE EXISTS (
SELECT *
FROM TITLEAUTHORS TA, TITLES T
WHERE TA.title_id=T.title_id AND TA.au_id=A.au_id AND T.price<10);
```



```
SELECT au_fname, au_lname
FROM AUTHORS A
WHERE EXISTS (
SELECT *
FROM TITLEAUTHORS TA, TITLES T
WHERE TA.title_id=T.title_id AND TA.au_id=A.au_id AND T.price<10)
```

#	AU_FNAME	AU_LNAME
1.	Marjorie	Green
2.	Albert	Ringer
3.	Anne	Ringer
4.	Michel	DeFrance
5.	Chastity	Locksley

2.2.4.11. Getting Information from the System Catalog

- *DESCRIBE Command*

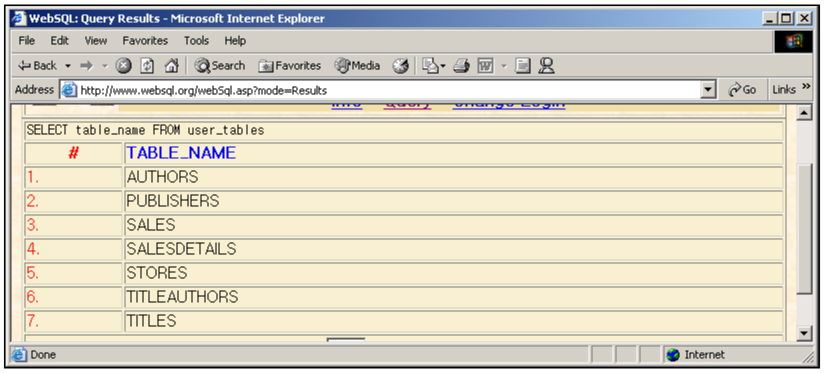
The DESCRIBE (or DESC) command gives you a quick summary of the table and all its columns.

```
DESC TITLES;
```

Note: WebSQL does not support DESCRIBE command.

```
SELECT * FROM tab;
SELECT table_name FROM user_tables;
```

Q24. Listing the tables you own:

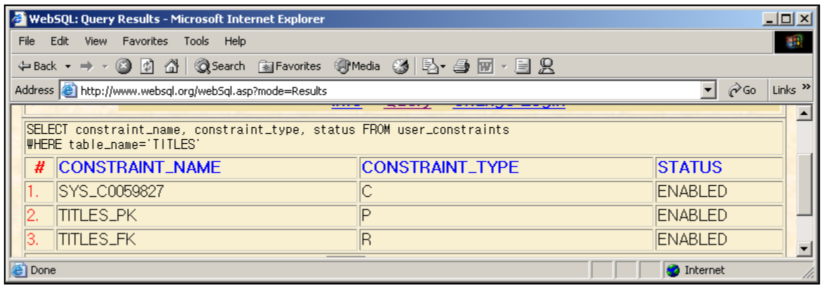


```
SELECT table_name FROM user_tables
```

#	TABLE_NAME
1.	AUTHORS
2.	PUBLISHERS
3.	SALES
4.	SALESDETAILS
5.	STORES
6.	TITLEAUTHORS
7.	TITLES

Q25. Listing constraints associated with the table you have permissions on

```
SELECT constraint_name, constraint_type, status FROM user_constraints
WHERE table_name='TITLES';
```


WebSQL: Query Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.websql.org/webSql.asp?mode=Results
Done
Internet

```
SELECT constraint_name, constraint_type, status FROM user_constraints
WHERE table_name='TITLES'
```

#	CONSTRAINT_NAME	CONSTRAINT_TYPE	STATUS
1.	SYS_C0059827	C	ENABLED
2.	TITLES_PK	P	ENABLED
3.	TITLES_FK	R	ENABLED

2.2.5. Creating Views

A view is a “virtual table” often used to simplify query commands, and also for security. Once created, you can refer to a view just as you would a table. The view does not really contain data – it merely looks it up for you every time you access the view. The data for each view is brought into the structure only when needed. Before creating a VIEW ensure that all the tables that are going to be defined as part of the view are available and are accessible. The syntax for the **CREATE VIEW** command is:

```
CREATE VIEW [(<column_list>)] AS <QUERY>
```

Suppose for e.g. we referred to publisher statistics very often. *Rather than run the following query every time…*

```
SELECT TITLES.pub_id, sum(price * qty_ordered) as total_sales, max(price * qty_ordered) as
max_sales, min(price * qty_ordered) as min_sales
FROM TITLES, SALESDETAILS
WHERE TITLES.title_id = SALESDETAILS.title_id
GROUP BY TITLES.pub_id;
```

We could create a view as follows:

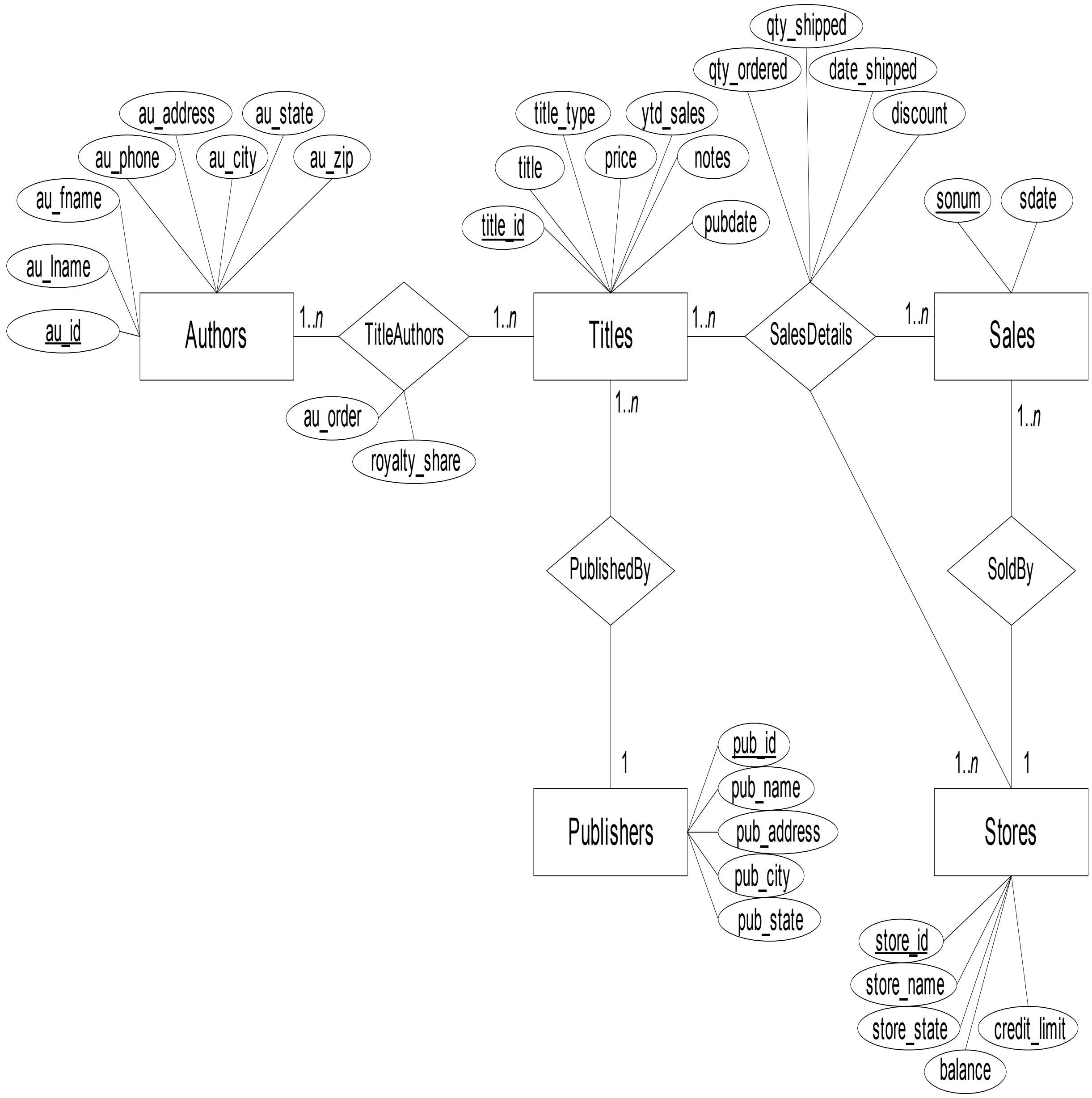
```
CREATE VIEW PUB_STATS As (
SELECT TITLES.pub_id, sum(price * qty_ordered) as total_sales, max(price * qty_ordered) as max_sales,
min(price * qty_ordered) as min_sales
FROM TITLES, SALESDETAILS
WHERE TITLES.title_id = SALESDETAILS.title_id
GROUP BY TITLES.pub_id);
```

APPENDIX 1: DDL for Constructing Publisher Schema

```
DROP TABLE AUTHORS CASCADE CONSTRAINTS;

CREATE TABLE AUTHORS (
       au_id                 CHAR(11) NOT NULL,
       au_lname              VARCHAR2(40),
       au_fname              VARCHAR2(20),
       au_phone              CHAR(12),
       au_address            VARCHAR2(40),
       au_city               VARCHAR2(20),
       au_state              CHAR(2),
       au_zip                CHAR(5),
       CONSTRAINT authors_pk  PRIMARY KEY (au_id)
);

DROP TABLE PUBLISHERS CASCADE CONSTRAINTS;

CREATE TABLE PUBLISHERS (
       pub_id                CHAR(4) NOT NULL,
       pub_name              VARCHAR2(40),
       pub_address           VARCHAR2(40),
       pub_city              VARCHAR2(20),
       pub_state             CHAR(2),
       CONSTRAINT publishers_pk     PRIMARY KEY (pub_id)
);

DROP TABLE TITLES CASCADE CONSTRAINTS;

CREATE TABLE TITLES (
       title_id              CHAR(6) NOT NULL,
       pub_id                CHAR(4),
       title                 VARCHAR2(80),
       title_type            VARCHAR2(20),
       price                 NUMBER(6,2),
       ytd_sales             NUMBER(10,2),
       notes                 VARCHAR2(200),
       pubdate               DATE,
       CONSTRAINT titles_pk   PRIMARY KEY (title_id),
       CONSTRAINT titles_fk   FOREIGN KEY (pub_id) REFERENCES PUBLISHERS
);

DROP TABLE TITLEAUTHORS CASCADE CONSTRAINTS;

CREATE TABLE TITLEAUTHORS (
       au_id                       CHAR(11) NOT NULL,
       title_id                    CHAR(6) NOT NULL,
       au_ord                      NUMBER,
       royalty_share               NUMBER(4,2),
       CONSTRAINT t_authors_pk     PRIMARY KEY (au_id, title_id),
       CONSTRAINT t_authors_fk1    FOREIGN KEY (au_id) REFERENCES AUTHORS,
       CONSTRAINT t_authors_fk2    FOREIGN KEY (title_id) REFERENCES TITLES
);
```

```
DROP TABLE STORES CASCADE CONSTRAINTS;

CREATE TABLE STORES (
      store_id                CHAR(4) NOT NULL,
      store_name              VARCHAR2(20),
      store_state             CHAR(2),
      balance                 NUMBER(10,2),
      credit_limit            NUMBER(10,2),
      CONSTRAINT stores_pk    PRIMARY KEY (store_id)
);

DROP TABLE SALES CASCADE CONSTRAINTS;

CREATE TABLE SALES (
      sonum                   NUMBER NOT NULL,
      store_id                CHAR(4),
      sales_date              DATE,
      CONSTRAINT sales_pk     PRIMARY KEY (sonum, store_id),
      CONSTRAINT sales_fk     FOREIGN KEY (store_id) REFERENCES STORES
);


DROP TABLE SALESDETAILS CASCADE CONSTRAINTS;

CREATE TABLE SALESDETAILS (
      sonum                   NUMBER NOT NULL,
      title_id                CHAR(6),
      store_id                CHAR(4),
      qty_ordered             NUMBER,
      qty_shipped             NUMBER,
      date_shipped            DATE,
      discount                NUMBER,
      CONSTRAINT s_details_pk       PRIMARY KEY (sonum, title_id, store_id),
      CONSTRAINT s_details_fk1      FOREIGN KEY (sonum, store_id) REFERENCES
SALES,
      CONSTRAINT s_details_fk2      FOREIGN KEY (title_id) REFERENCES TITLES
);
```

APPENDIX 2: ER Diagram for Publisher Schema


qty_shipped
qty_ordered
date_shipped
discount
au_address
au_state
au_phone
au_city
au_zip
au_fname
au_lname
au_id
title_type
ytd_sales
title
price
notes
title_id
pubdate
sonum
sdate
Authors
TitleAuthors
Titles
SalesDetails
Sales
1..n
1..n
1..n
1..n
1..n
1..n
au_order
royalty_share
PublishedBy
SoldBy
1
1..n
1
Publishers
Stores
pub_id
pub_name
pub_address
pub_city
pub_state
store_id
store_name
store_state
credit_limit
balance