SQL

WHAT IS SQL?
According to Wiki, "SQL (Structured Query Language) (pronounced see-qual), is a database computer language designed for the retrieval and management of data in relational database management systems (RDBMS), database schema creation and modification, and database object access control management."

WHAT DOES SQL DO?
SQL is the language of the databases. All modern relational databases (Oracle, MS-SQL, MY-SQL) now use SQL to provide the users with an interface to manage databases.

WHAT IS A RELATIONAL DATABASE?
Relational databases present the data to the users as tables and allow users to modify the data using operators. A relational database should adhere to **Codd's 12 rules** (actually 13 rules numbered 0 to 12). *(Read more about Codd's 12 rules at wikipedia.org)*

WHAT IS AN SQL STATEMENT?

An SQL Statement is a statement written in SQL. An SQL statement can be classified in to 4 groups. These include:

1. Data Definition Statements
2. Data Manipulation and Retrieval Statements
3. Data Access Control Statements
4. Data Transaction Control Statement

BASICS OF SQL

Simply speaking, SQL can be viewed as a language consisting of three different elements.

1. **Data types** that allow storing of data in specific formats such as text or numbers or dates
2. **Operators** that allow us to slice and dice the data when used inside an SQL statement such as MAX, COUNT etc. and
3. **SQL statements** that allow the creation of databases & tables, modification of data in those tables, granting or revoking user permissions and managing transactions.

LIST OF SQL DATA TYPE

Please note that the data types may differ based on the database used.

Text Types:

Data type	Description
CHAR (size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR (size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. **Note:** If you put a greater value than 255 it will be converted to a TEXT type
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT	Holds a string with a maximum length of 65,535 characters
BLOB	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM (x,y,z,etc.)	Let you enter a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted.
	You enter the possible values in this format: ENUM('X','Y','Z')
SET	Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice

Number types:

Data type	Description
TINYINT (size)	-128 to 127 normal. 0 to 255 UNSIGNED*. The maximum number of digits may be specified in parenthesis
SMALLINT (size)	-32768 to 32767 normal. 0 to 65535 UNSIGNED*. The maximum number of digits may be specified in parenthesis

MEDIUMINT (size)	-8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*. The maximum number of digits may be specified in parenthesis
INT (size)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis
BIGINT (size)	-9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615 UNSIGNED*. The maximum number of digits may be specified in parenthesis
FLOAT (size,d)	A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DOUBLE (size,d)	A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DECIMAL (size,d)	A DOUBLE stored as a string, allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

*The integer types have an extra option called UNSIGNED. Normally, the integer goes from an negative to positive value. Adding the UNSIGNED attribute will move that range up so it starts at zero instead of a negative number.

Date types:

Data type	Description
DATE()	A date. Format: YYYY-MM-DD
	Note: The supported range is from '1000-01-01' to '9999-12-31'
DATETIME()	*A date and time combination. Format: YYYY-MM-DD HH:MM:SS
	Note: The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP()	*A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MM:SS
	Note: The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC
TIME()	A time. Format: HH:MM:SS
	Note: The supported range is from '-838:59:59' to '838:59:59'

YEAR()	A year in two-digit or four-digit format.
	Note: Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

Let's use the following table as an example:
EmpList

Name	EmpDateofJoin	EmpID	DeptID	Experience
John	23-Jan-77	1903	24	12
Terri	18-Aug-65	1002	9	23
Mark	1-Apr-83	1008	9	4
Lanni	9-Jan-80	1321	9	12
Kate	31-Apr-69	1890	6	35
Bruce	29-Jun-83	1607	4	15

LIST OF SQL OPERATORS AND FUNCTIONS

Operator	Description
WHERE	Specifies a condition to be used to narrow down the number of records selected. Example: `SELECT * FROM EmpList WHERE Experience > 10;`
AND	Evaluates two or more conditions to generate a list of records that match each one (ALL) of the criteria. Example to select all records where Experience is greater than 10 and EmployeeID less than 1500, use: `SELECT * FROM EmpList WHERE (Experience > 10 AND EmpID > 10);`
OR	Evaluates two or more conditions to generate a list of records that match any one of the criteria. Example to select all records where Experience is greater than 10 OR EmployeeID less than 1500, use: `SELECT * FROM EmpList WHERE (Experience > 10 OR EmpID > 10);`
IN	Provides a specific set of values to choose from. Example, to select records where the name is John or Mark or Kate, use: `SELECT * FROM EmpList WHERE Name IN ("John", "Mark" "Kate");`
NOT	Includes the records not meeting the condition. Example to select all records where Experience is NOT greater than 10, use: `SELECT * FROM EmpList WHERE NOT (Experience> 10);`

HAVING	Used in conjunction with GROUP BY caluse.
	`SELECT Name, Employee ID`
	`FROM EmpList`
	`GROUP BY DeptID`
	`HAVING UnitsSold> 20;`
	
	(Also see data aggregation operator list -> Group By)

BETWEEN	Provides a range to choose from. Example, to select records where the Experience is greater than 12 and less than 20, use: `SELECT * FROM EmpList WHERE Experience BETWEEN 12 AND 20;`
AS	Provides a way of temporarily storing data. Also known as an Alias. Example: `SELECT Name AS NEWNAME FROM EmpList;` will return all records and also label the column as NEWNAME. Can also be used within a procedure.

LIST OF SQL DATA AGGREGATION OPERATORS AND FUNCTIONS

Operator	Description
MAX	Selects the record with the highest value from all the records selected by the query. Example: `SELECT MAX (Experience) FROM EmpList;`
MIN	Selects the record with the lowest value from all the records selected by the query. Example: `SELECT MAX (Experience) FROM EmpList;`
SUM	Shows the sum of the column for all the records selected by the query. Example: `SELECT SUM(Experience) FROM EmpList;`
AVERAGE	Shows the average of the column for all the records selected for the query. Example: `SELECT AVERAGE (Experience) FROM EmpList;`

DISTINCT	Selects only the UNIQUE records from a table. Example to generate a list of people none of whom have a repeating name use:
	
	`SELECT UNIQUE Name FROM EmpList;`
	
	Can also be used with multiple columns. Example, to generate a list of people none of whom have a repeating name or DepartmentID, use:
	
	`SELECT UNIQUE Name, DeptID FROM EmpList;`

COUNT	Shows the overall number of records in the table of the column selected for the query. Example to get the number of records in where there is a Name in the table: `SELECT COUNT(Name) FROM EmpList;`
JOIN	Used to select data from two or more tables based on the relationship between those columns. Joins work like a recursion. **Each** record in the first table is matched with all the records in the second table to check if the records has a match as per the criteria specified. There are two types of joins: INNER (works only when both tables have a matching value) and OUTER (works even when the value in one table does not match with any record in the second table)
INNER JOIN	The INNER JOIN keyword returns records only if there is a match in both of the tables. If a record in table1 does not have a match in

	table2, the record will not be displayed in the output.
	
	`SELECT column1, column2, column3…columnN FROM table1`
	`INNER JOIN table2`
	`ON table1.column1=table2.column2;`

OUTER JOIN	The outer join has 2 types: LEFT JOIN and RIGHT JOIN;

LEFT JOIN	The LEFT JOIN keyword returns records from table1 (left table) even when there is no match in table2. If a record in table1 does not have a match in table2, the record from table1will still be displayed in the output alongwith with all the regular matches.
	
	`SELECT column1, column2, column3…columnN FROM table1`
	`LEFT JOIN table2`
	`ON table1.column1=table2.column2;`

RIGHT JOIN	The RIGHT JOIN keyword returns records from table2 (right table) even when there is no match in table1. If a record in table2 does not have a corresponding match in table1, the record from table2 will still be displayed in the output alongwith with all the regular matches.
	
	`SELECT column1, column2, column3…columnN FROM table1`
	`RIGHT JOIN table2`
	`ON table1.column1=table2.column2;`

FULL JOIN	The FULL JOIN keyword returns records from table1 (left table) and table2 even if there are no matches. If a record in table1 does not have a corresponding match in table2 OR vice-versa, the records will still be displayed alongwith with all the regular matches. In some sense it is a combination of two separate inner join statements, one inner join of table1 on table2 and the other inner join of table2 on table1, excluding duplicate records.
	
	`SELECT column1, column2, column3…columnN FROM table1`
	`FULL JOIN table2`
	`ON table1.column1=table2.column2;`

UNION	Provide a convenient way of retrieving data from more than one table by joining two or more queries together. It saves the time required in running separate queries if the same set of fields need are stored in different tables, say monthly sales data stored in tables that are created at the starting of each month and store data pertaining to that month only. Example, to fetch all records from three different tables which store the sales data for JAN, FEB and MARCH, use:
	
	`SELECT * FROM salesdata_JAN WHERE salesvol>20`
	`UNION`

	SELECT * **FROM** salesdata_FEB **WHERE** salesvol>20 **UNION** **SELECT** * **FROM** salesdata_MAR **WHERE** salesvol>20;
DESCRIBE	A general statement used to describe an object. An object can be a table, database, view, procedure etc. **DESC** tablename; **DESC** viewname; **DESC** procedurename; **DESC** functionname; **DESC** packagename; **DESC** objectname;
GROUP BY	GROUP BY forms Groups based on the repeating values in the column(s) specified and returns one record for each such group. The end result is a set of records in which the values (of the column specified in the GROUP BY condition) are non-repeating and are unique. It also sorts the resulting set of records. It will also retrieve a DISTINCT RECORD SET. So using GROUP BY on “DeptID” would yield only one record per department. **SELECT** Name, EmpID **FROM** EmpList **GROUP BY** DeptID;
ORDER BY	Simply sorts the records by the columns included in the ORDER BY statement. If there are more than two records with the same value for that column (two employees with the same Dept), it will show two or more records. It does not show DISTINCT records, it shows ALL records, only the output is sorted. To see all the records in an ascending order by Name use: **SELECT** * **FROM** tablename **ORDER BY** Name;
TOP	Selects the first N number of records from the table as per the criteria. The number of records to be retrieved can be specified as an absolute number or a percentage. **SELECT TOP** 10 * **FROM** EmpList; **SELECT TOP** 25 **PERCENT** * **FROM** EmpList;
STDEV	Provides the standard deviation of the records
FIRST	Returns the value of the first record for the specified column
LAST	Returns the value of the last record for the specified column

LIST OF SQL TEXT AGGREGATION OPERATORS AND FUNCTIONS

Operator	Description
UCASE	Converts a field to upper case To select Names converted to uppercase in the EmpList table use: `SELECT UCASE(Name) FROM EmpList;`
LCASE	Converts a field to lower case To select Names converted to lowercase in the EmpList table use: `SELECT LCASE(Name) FROM EmpList;`
MID (c,start, [end])	Extract characters from a text field
LEN	Returns the length of a text field To select the length of Names in the EmpList table use: `SELECT LEN(Name) FROM EmpList;`
INSTR (c,char)	Returns the numeric position of a named character within a text field
LEFT (c,number_of _char)	Return the left part of a text field requested To select the first 5 chars from Name in the EmpList table use: `SELECT LEFT(Name,5)FROM EmpList;`
RIGHT (c,number_of _char)	Return the right part of a text field requested To select the last 5 chars from Name in the EmpList table use: `SELECT RIGHT(Name,5)FROM EmpList;`
ROUND (c,decimals)	Rounds a numeric field to the number of decimals specified
MOD (x,y)	Returns the remainder of a division operation
NOW()	Returns the current system date

FORMAT (c,format)	Changes the way a field is displayed
	
	For example to convert Name to lowercase use
	`SELECT FORMAT(Name, "<") FROM EmpList;`
	
	For example to convert Name to uppercase use
	`SELECT FORMAT(Name, ">") FROM EmpList;`
	
	For example to convert Experience to decimal use
	`SELECT FORMAT(Experience, "##,##0.00") FROM EmpList;`

DATEDIFF (d,date1,date 2)	Used to perform date calculations To calculate the number of days since the employee joined the organization use:

	`Select` **`DateDiff(`**`"d", Now, EmpDateofJoin`**`)`** **`FROM`** `EmpList;` to get the # of days `Select` **`DateDiff(`**`"w", Now, EmpDateofJoin`**`)`** **`FROM`** `EmpList;` to get the # of weeks

SQL STATEMENTS

SQL statements can be classified into four basic categories.

1. **Data Definition Statements**
   These elements allow the user to define the overall structure of the data and how it is organized. (CREATE, ALTER, DROP)

2. **Data Manipulation and Retrieval Queries**
   These elements allow the user to work with the data once it has been defined. Using these elements a user can view and modify data stored in a table(s) (SELECT, INSERT, UPDATE, DELETE)

3. **Data Access Control Statements**
   These elements allow a system administrator to give and take away various permissions to and from the users of the database. (GRANT, REVOKE)

4. **Data Transaction Control Statements**
   These elements provide a way of controlling the state of the database and a particular transaction. These are similar to the save command in windows and allow you to save the “work” done so far. They allow you to resume from the last saved point onwards rather than having to start again at the beginning. (COMMIT, ROLLBACK, SAVEPOINT)

SQL DATA DEFINITION STATEMENTS

Please note that all words appearing in bold are language constructs. They are not variables and therefore should not be modified.

SQL: CREATE

USED FOR

Creating a database, table, Index or roles

HOW TO WRITE THE SQL CREATE STATEMENT

For creating a database:

```
CREATE DATABASE databasename
```

For creating a table:

```
CREATE TABLE tablename
(
Datafield_1 data_type,
Datafield_2 data_type,
...
Datafield_n data_type
)
```

For creating an role:

```
CREATE ROLE role [IDENTIFIED BY password];
```

For creating an index:

```
CREATE UNIQUE INDEX indexname ON tablename (column_name)
```

EXAMPLES

To create a database by the name of "**EmployeeRecords"**

```
CREATE database EmployeeRecords;
```

To create a table by the name "**EmpList"** with columns "EmpName", "EmpDateofJoin", "EmpID", "DeptID"s

```
CREATE TABLE EmpList
(
Name varchar,
EmpDateofJoin date,
EmpID varchar,
DeptID varchar
)
```

For creating an role called “regularuser”:

```
CREATE ROLE regularuser [IDENTIFIED BY mike$123@];
```

To create an Index by the name “**EmpIndex”** on column “Name” and “DeptID” in the table “EmpList”

```
CREATE INDEX EmpIndex
ON EmpList (Name, DeptID)
```

SQL: ALTER

USED FOR
Adding, dropping and modify table columns
Adding and dropping constraints
Enabling and Disabling constraints

HOW TO WRITE THE SQL ALTER STATEMENT

For adding a column:

```
ALTER TABLE table_name ADD column_name datatype;
```

For deleting a column:

```
ALTER TABLE table_name DROP column_name;
```

For adding a constraint:

```
ALTER TABLE table_name ADD CONSTRAINT
newconstraint_definition
```

For deleting a constraint:

```
ALTER TABLE table_name DROP CONSTRAINT constraint
```

EXAMPLES

For adding a column "**EmpDateofBirth**" to table "EmpList":

```
ALTER TABLE EmpList ADD EmpDateofBirth date;
```

For deleting a column "**EmpDateofBirth**" to table "EmpList":

```
ALTER TABLE EmpList DROP EmpList;
```

For adding a constraint (A new table "SalesTeam" is being created and we want it to have only those employees that are in the "EmpList" table)

EmpList

Name	EmpDateofJoin	EmpID	DeptID

John	23-Jan-77	1903	24
Terri	18-Aug-65	1002	9

Sales

Name	SalesEmpID	UnitsSold	DateofSale
John	1903	4	1-Jan-08
Terri	1002	6	1-Jan-08
Terri	1002	9	1-Jan-08
John	1903	9	2-Jan-08
John	1903	9	3-Jan-08
John	1903	3	4-Jan-08
Terri	1002	2	4-Jan-08
Terri	1002	8	5-Jan-08
Terri	1002	15	5-Jan-08
Terri	1002	1	5-Jan-08

```
ALTER TABLE Sales ADD CONSTRAINT validemployee FOREIGN KEY
(SalesEmpID) REFERENCES EmpList (EmpID);
```

For deleting a constraint:

```
ALTER TABLE EmpList DROP CONSTRAINT validemployee;
```

SQL: DROP

USED FOR
Dropping index, table, database or role

HOW TO WRITE THE SQL DROP STATEMENT

For dropping an index:

```
DROP INDEX index_name ON table_name;
```

For dropping a table:

```
DROP TABLE table_name;
```

For dropping a database:

```
DROP DATABASE database_name;
```

For dropping a role:

```
DROP ROLE roll;
```

EXAMPLES

For dropping an index called **EmpIndex** from a table called EmpList:

```
DROP INDEX EmpIndex ON EmpList;
```

For dropping a table called **EmpList**:

```
DROP TABLE EmpList;
```

For dropping a database called **EmployeeRecords**:

```
DROP DATABASE EmployeeRecords;
```

For dropping a role called regularuser:

```
DROP ROLE regularuser;
```

SQL DATA MODIFICATION STATEMENTS

SQL: SELECT

USED FOR

Retrieving records from one or many tables based on a certain criteria. This is one of the most useful and frequently used statements in SQL.

HOW TO WRITE THE SQL SELECT STATEMENT

For selecting all records from a table:

```
SELECT * FROM table_name;
```

For selecting a certain set of columns from a table:

```
SELECT Column1, Column2, Column3, ... ColumnN  FROM
table_name;
```

For selecting a certain set of columns from a table based on a specific criteria:

```
SELECT Column1, Column2, Column3, ... ColumnN  FROM
table_name WHERE condition ;
```

The query will select all records for which the criteria for the condition are met.

For selecting a certain set of columns from a table based on a multiple criteria:

```
SELECT Column1, Column2, Column3, ... ColumnN
FROM table_name
WHERE (condition1 AND condition2)
GROUP BY expression1, expression2
HAVING expression
ORDER BY Column1, Column2;
```

The query will select all records for which the criteria for the conditions are met and as per the options specified in “GROUP BY” and “ORDER BY” expressions.

EXAMPLES

EmpList

Name	EmpDateofJoin	EmpID	DeptID
John	23-Jan-77	1903	24
Terri	18-Aug-65	1002	9
Mark	1-Apr-83	1008	9
Terri	9-Jan-80	1321	9
Linda	31-Apr-69	1890	6
Terri	29-Jun-83	1607	4

For selecting all records from a table called EmpList:

```
SELECT * FROM EmpList;
```

For selecting a certain set of columns from a table:

```
SELECT Name, EmpID, DeptID FROM EmpList;
```

will give the following result:

Name	EmpID	DeptID
John	1903	24
Terri	1002	9
Mark	1008	9
Terri	1321	9
Linda	1890	6
Terri	1607	4

For selecting a certain set of columns from a table based on a specific criteria:

```
SELECT Name, EmpID, DeptID FROM EmpList WHERE DeptID = 9;
```

Will give the following result:

Name	EmpID	DeptID
Terri	1002	9

For selecting a certain set of columns from a table based on a multiple criteria:

```
SELECT Name, EmpID, DeptID
FROM EmpList
WHERE (DeptID = 4 OR DeptID = 6 OR DeptID = 9)
ORDER BY DeptID;
```

Will give the following records:

Name	EmpID	DeptID
Terri	1607	4
Linda	1890	6
Terri	1002	9
Mark	1008	9
Terri	1321	9

Point to Note

ORDER BY sorts the records using the columns included in the ORDER BY statement. If there are more than two records with the same value for that column (two employees with the same Dept), it will show two or more records. It does not show DISTINCT records.

GROUP BY forms Groups (of course) and sorts but this means it also SORTS; but it will also retrieve a DISTINCT RECORD SET. So using GROUP BY on “DeptID” would yield only one record per department

Which columns are selected and are shown as output is however, not determined by either ORDER BY or GROUP BY expressions. It is simply determined by the list of columns given just after the SELECT clause.

The SQL HAVING clause is used in conjunction with the GROUP BY clause to specify a search condition for a group or aggregate. The HAVING clause is similar in nature to the WHERE clause, but is applicable to groups. In contrast the WHERE clause is applied to individual rows, not to groups.

For selecting a certain set of columns from a table based on a multiple criteria:
Sales

Name	EmpID	UnitsSold	DateofSale
Terri	1002	4	1-Jan-08
Linda	1890	6	1-Jan-08
Terri	1002	9	1-Jan-08
Mark	1008	9	2-Jan-08
Terri	1321	9	3-Jan-08
Terri	1607	3	4-Jan-08
Linda	1890	2	4-Jan-08
Terri	1002	8	5-Jan-08
Mark	1008	15	5-Jan-08
Terri	1321	1	5-Jan-08

```
SELECT Name, UnitsSold
FROM Sales
GROUP BY Name
HAVING UnitsSold> 20
ORDER BY Name;
```

Will give the following records:

Name	UnitsSold
Mark	24
Terri	34

Even though Terri appears before Mark in the “Sales” table, due to the “ORDER BY” clause which has the expression as Name, Mark appears before Terri in the output.

SQL: INSERT

USED FOR
Inserting a set of records into a table.

HOW TO WRITE THE SQL INSERT STATEMENT

For inserting one COMPLETE record:

```
INSERT INTO tablename VALUES (value1, value2, value3…);
```

For inserting one record with field level mapping:

```
INSERT INTO tablename (Column1, Column2, Column3, Column4)
VALUES (value1, value2, value3, value4);
```

Inserting ALL records from one table to another:

```
SELECT * INTO tablename1 FROM tablename2;
```

(This is a slight variation where the records are inserted directly from one tale into another table.)

Inserting more than one record from one table to another:

```
INSERT INTO tablename1 (Column1, Column2, Column3, Column4)
SELECT Column11, Column22, Column33, Column44
FROM tablename2
WHERE condition;
```

EXAMPLES

Sales

Name	EmpID	UnitsSold	DateofSale
Terri	1002	4	1-Jan-08
Linda	1890	6	1-Jan-08
Terri	1002	9	1-Jan-08
Mark	1008	9	2-Jan-08

To insert a new record into the above table:

```
INSERT INTO Sales VALUES ('Mark', 1008, 6,'3-Jan-08');
```

For inserting one record with field level mapping:

```
INSERT INTO Sales (Name, EmpID, UnitsSold, DateofSale)
VALUES ('Mark', 1008, 6,'3-Jan-08');
```

Moving the entire set of records from one table to another, say moving sales data from January sales table to Quarter1 sales table:

```
SELECT * INTO salesquarter1 FROM SalesJAN;
```

Inserting more than one record from one table to another:

```
INSERT INTO salesquarter1 (Name, EmpID, UnitsSold,
DateofSale)
SELECT Name, EmpID, UnitsSold, DateofSale
FROM SalesJAN;
WHERE UnitsSold > 0;
```

SQL: UPDATE

USED FOR

Modifying the existing data in a table.

HOW TO WRITE THE SQL INSERT STATEMENT

For updating (modifying) ALL records by changing the value of a column:

```
UPDATE tablename SET column = value;
```

For updating (modifying) records as per specific criteria by changing the value of more than one column:

```
UPDATE tablename
SET column1 = value1, column1 = value2
WHERE condition;
```

EXAMPLES

Sales

Name	EmpID	UnitsSold	DateofSale
Terri	1002	4	1-Jan-08
Linda	1890	6	1-Jan-08
Terri	1002	9	1-Jan-08
Mark	1008	9	2-Jan-08

To modify the Sales table and set Name to Terri Smith where the existing Name is Terri (Change in name after marriage):

```
UPDATE Sales
SET Name = "Terri Smith"
WHERE Name="Terri";
```

Will make the following changes:

Sales

Name	EmpID	UnitsSold	DateofSale
Terri Smith	1002	4	1-Jan-08
Linda	1890	6	1-Jan-08
Terri Smith	1002	9	1-Jan-08
Mark	1008	9	2-Jan-08

SQL: DELETE

USED FOR
Deleting records from a table

HOW TO WRITE THE SQL DELETE STATEMENT

For deleting all records:

```
DELETE * tablename;
```

An associated command TRUNCATE can also be used to delete all rows from the table. The advantage is that TRUNCATE reduces the size of the table after deleting the records thus freeing up disk space.

For deleting all records:

```
DELETE FROM tablename WHERE condition;
```

EXAMPLES

Sales

Name	EmpID	UnitsSold	DateofSale
Terri	1002	4	1-Jan-08
Linda	1890	6	1-Jan-08
Terri	1002	9	1-Jan-08
Mark	1008	9	2-Jan-08

For dropping all records where the Employee ID is 1002 use:

```
DELETE FROM Sales WHERE EmpID = 1002;
```

Will result in:

Sales

Name	EmpID	UnitsSold	DateofSale
Linda	1890	6	1-Jan-08
Mark	1008	9	2-Jan-08

SQL: TRUNCATE

USED FOR

Deleting all records from a table and compacting the table by reducing its size

HOW TO WRITE THE SQL DELETE STATEMENT

For deleting all records:

```
TRUNCATE TABLE tablename;
```

EXAMPLES

Sales

Name	EmpID	UnitsSold	DateofSale
Terri	1002	4	1-Jan-08
Linda	1890	6	1-Jan-08
Terri	1002	9	1-Jan-08
Mark	1008	9	2-Jan-08

For dropping all records where the Employee ID is 1002 use:

```
TRUNCATE TABLE Sales;
```

TRUNCATE deletes all the records from a table and reduces its size. It however does not delete the table itself.

SQL DATA ACCESS CONTROL STATEMENTS

SQL: GRANT

USED FOR
Used for providing access and privileges to users on various *objects* in the database.

Other elements used in conjunction with the GRANT Command
OBJECT
An object is a set of defined entities in SQL. They include tables, procedures, views etc. The GRANT command uses the object as an entity on which a user has to be provided a certain right or permission.

PRIVILEGE
A privilege is the ability to run any of data definition or data modification statements such as CREATE, ALTER, DROP, SELECT, INSERT, UPDATE, DELETE, ALL etc. on the database.

ROLES
A set of privileges clubbed together forms a role. Rather than having to use multiple GRANT statements to provide a specific set of permissions to a large set of users, the database administrator can simply club those privileges into a ROLE and then GRANT them. Depending on the type of database, some roles may already be defined such as the DBA (database admin) and RESOURCE (normal users)

PUBLIC
This is a keyword representing all users.

WITH GRANT
Used at the end of the GRANT statement Indicates whether the user has to right to further GRANT this permission to anybody else or not. USE WITH CUTION. A user once granted this privilege can further GRANT this to other users. However at a later time when those permissions are revoked from the first user, the revoking of the privilege does NOT automatically propagate to all users who were granted privileges by the first user.

HOW TO WRITE THE SQL GRANT STATEMENT

For granting access to an object:

```
GRANT privilege
ON object
TO {user |PUBLIC |role}
[WITH GRANT OPTION];
```

EXAMPLES

Sales

Name	EmpID	UnitsSold	DateofSale
Terri	1002	4	1-Jan-08
Linda	1890	6	1-Jan-08
Terri	1002	9	1-Jan-08
Mark	1008	9	2-Jan-08

For granting access to a user mark to run a select query and also allowing him to pass this right to other users use:

```
GRANT select
ON Sales
TO mark
with grant option;
```

SQL: REVOKE

USED FOR

Revoking access on a particular object from a user

HOW TO WRITE THE SQL REVOKE STATEMENT

For granting access to an object:

```
REVOKE privilege
ON object
FROM {user_name |PUBLIC |role_name};
```

EXAMPLES

Sales

Name	EmpID	UnitsSold	DateofSale
Terri	1002	4	1-Jan-08
Linda	1890	6	1-Jan-08
Terri	1002	9	1-Jan-08
Mark	1008	9	2-Jan-08

For revoking access granted to a user mark to run a select query:

```
REVOKE select
ON Sales
FROM mark;
```

SQL DATA TRANSACTION CONTROL STATEMENTS

SQL: SAVEPOINT

USED FOR

Used to create a ‘milestone’ that the database recognizes as a last saved point. If something goes wrong while running a query or otherwise, this can help the system go back to a particular savepoint.

HOW TO WRITE THE SQL SAVEPOINT STATEMENT

For creating a particular savepoint that the system recognizes as a fall-back point in case something goes wrong:

```
SAVEPOINT savepoint;
```

EXAMPLES

After all the transactions for the day are complete, creating a savepoint for the day:

```
SAVEPOINT EndOfDay22JAN08;
```

SQL: ROLLBACK

USED FOR

Rolling back the database to a previous savepoint and mitigating any changes made since that point

HOW TO WRITE THE SQL ROLLBACK STATEMENT

A general rollback:

```
ROLLBACK;
```

Rollback to a particular point:

```
ROLLBACK TO SAVEPOINT savepoint;
```

EXAMPLES

A query accidentally made changes to one of the tables on the database. You want to rollback to a previous savepoint :

```
ROLLBACK TO SAVEPOINT EndOfDay22JAN08;
```

SQL: COMMIT

USED FOR

Committing a set of changes permanently to the database and making them visible to other users.

HOW TO WRITE THE SQL COMMIT STATEMENT

Committing work done so far to the database and making those changes permanent:

```
COMMIT;
```

That's it !

EXAMPLES

You just ran a query inserting a set of records to in a particular table:

```
COMMIT;
```

Will make those changes permanent

LIST OF SQL STATEMENTS

SQL: CREATE
Used for creating a database, table, Index or roles

SQL: ALTER
Used for adding, dropping and modify table columns, adding and dropping constraints, enabling and Disabling constraints

SQL: DROP
Used for dropping index, table, database or role

SQL: SELECT
Used for retrieving records from one or many tables based on a certain criteria. This is one of the most useful and frequently used statements in SQL.

SQL: INSERT
Used for inserting a set of records into a table.

SQL: UPDATE
Used for modifying the existing data in a table.

SQL: DELETE
Used for deleting records from a table

SQL: TRUNCATE
Used for deleting all records from a table and compacting the table by reducing its size

SQL: GRANT
Used for providing access and privileges to users on various objects in the database.

SQL: REVOKE
Used for revoking access on a particular object from a user

SQL: SAVEPOINT
Used to create a 'milestone' that the database recognizes as a last saved point. If something goes wrong while running a query or otherwise, this can help the system go back to a particular savepoint.

SQL: ROLLBACK
Used for rolling back the database to a previous savepoint and mitigating any changes made since that point

SQL: COMMIT
Used for committing a set of changes permanently to the database and making them visible to other users