## ***Insert***

To insert a record into a table, you use the following syntax:

```
INSERT INTO <tablename>(<ColumnName>, <columnName>, ...)
VALUES(<value1>, <value2>, …)
```

## ***Updates***

Updates allow you to change existing records. The syntax is:

```
UPDATE <TableName>
SET <ColumnName> = <New Value>,
<ColumnName>=<new value>
WHERE <ColumnName> = <criteria>
```

## ***Deletes***

•Deletes allow you to remove a record from a table:

```
DELETE FROM <TableName>
WHERE <columnName> = <criteria>
```

Deletes and Updates

•Deletes and updates are dangerous. If you do not specify a criteria, the update or delete will be applied to all the rows in a table.

•Also, referential integrity may prevent a deletion. You cannot delete a parent that has children in another table.

### ***SubQuery Example***

```
SELECT DISTINCT COUNT(*) AS Total,
(SELECT COUNT(*)
FROM Session
WHERE SessionStatus='NS') AS NoShow,
(SELECT COUNT(*)
FROM Session
WHERE SessionStatus='c') AS Completed
FROM Session
```

This example shows subqueries used in the SELECT clause to return Aggregate values.

### ***Locating Duplicates***

```
SELECT Lastname, firstname, email, phone, COUNT(*) AS [duplicates]
FROM contact
GROUP BY Lastname, firstName, email, Phone
HAVING COUNT(*) >1
```

### ***Documentation: Testing Plans***

•When testing the database, you should document all your SQL queries and their results.

•On the next slide is a sample of a test table, showing the test and results.

### ***Union, Intersect, and Difference***

•Can use normal set operations of Union, Intersection, and Difference to combine results of two or more queries into a single result table.

•Union of two tables, A and B, is table containing all rows in either A or B or both.

•Intersection is table containing all rows common to both A and B.

•Difference is table containing all rows in A but not in B.

•Two tables must be union compatible.

•Format of set operator clause in each case is:

```
op [ALL] [CORRESPONDING [BY {column1 [, ...]}]]
```

•
•If CORRESPONDING BY specified, set operation performed on the named column(s).
•If CORRESPONDING specified but not BY clause, operation performed on common columns.
•If ALL specified, result can include duplicate rows.

***Use of UNION***

List all cities where there is either a branch office or a property.

```
  (SELECT city
  FROM Branch
  WHERE city IS NOT NULL) UNION
  (SELECT city
  FROM PropertyForRent
  WHERE city IS NOT NULL);
```

•Or

```
    (SELECT *
  FROM Branch
  WHERE city IS NOT NULL)
  UNION CORRESPONDING BY city
  (SELECT *
  FROM PropertyForRent
  WHERE city IS NOT NULL);
```

```
INSERT INTO Contact(LastName, FirstName, Email, Phone)
SELECT StudentLastName AS LastName,
StudentFirstName AS FirstName,
StudentEmail AS Email,
StudentPhone AS Phone
FROM Student
WHERE StudentEmail IS NOT NULL
UNION
SELECT LastName,
FirstName,
Email,
Phone
FROM
WHERE Email IS NOT NULL
```

This UNION query joins the tables Student and  into a single result and writes them to the table Contact.

***Use of INTERSECT***

List all cities where there is both a branch office and a property.

```
      (SELECT city FROM Branch)
  INTERSECT
  (SELECT city FROM PropertyForRent);
```

•Or

```
  (SELECT * FROM Branch)
  INTERSECT CORRESPONDING BY city
  (SELECT * FROM PropertyForRent);
```

•Could rewrite this query without INTERSECT operator:

```
  SELECT b.city
  FROM Branch b PropertyForRent p
  WHERE b.city = p.city;
```

•Or:

```
   SELECT DISTINCT city FROM Branch b
```

```
  WHERE EXISTS
  (SELECT * FROM PropertyForRent p
  WHERE p.city = b.city);
```

***Use of EXCEPT***

List of all cities where there is a branch office but no  properties.

```
   (SELECT city FROM Branch)
EXCEPT
(SELECT city FROM PropertyForRent);
```

•Or

```
(SELECT * FROM Branch)
EXCEPT CORRESPONDING BY city
(SELECT * FROM PropertyForRent);
```

•Could rewrite this query without EXCEPT:

```
  SELECT DISTINCT city FROM Branch
  WHERE city NOT IN
  (SELECT city FROM PropertyForRent);
```

•Or

```
  SELECT DISTINCT city FROM Branch b
  WHERE NOT EXISTS
  (SELECT * FROM PropertyForRent p
  WHERE p.city = b.city);
```

***INSERT***

```
   INSERT INTO TableName [ (columnList) ]
VALUES (dataValueList)
```

•columnList is optional; if omitted, SQL assumes a list of all columns in their original CREATE TABLE order.
•Any columns omitted must have been declared as NULL when table was created, unless DEFAULT was specified when creating column.
•dataValueList must match columnList as follows:
•number of items in each list must be same;
•must be direct correspondence in position of items in two lists;
•data type of each item in dataValueList must be compatible with data type of corresponding column.

Insert a new row into Staff table supplying data for all columns.

```
   INSERT INTO Staff
VALUES (‘SG16’, ‘Alan’, ‘Brown’, ‘Assistant’, ‘M’, Date‘1957-05-25’, 8300, ‘B003’);
```

Insert a new row into Staff table supplying data for all mandatory columns.

```
    INSERT INTO Staff (staffNo, fName, lName,
                                position, salary, branchNo)
VALUES (‘SG44’, ‘Anne’, ‘Jones’,
               ‘Assistant’, 8100, ‘B003’);
```

•Or

```
INSERT INTO Staff
VALUES (‘SG44’, ‘Anne’, ‘Jones’, ‘Assistant’, NULL,
                NULL, 8100, ‘B003’);
```

•Second form of INSERT allows multiple rows to be copied from one or more tables to another:

```
  INSERT INTO TableName [ (columnList) ]
  SELECT ...
```

Assume there is a table StaffPropCount that contains names of staff and number of properties they manage:

StaffPropCount(staffNo, fName, lName, propCnt)
Populate StaffPropCount using Staff and PropertyForRent tables.

```
INSERT INTO StaffPropCount
  (SELECT s.staffNo, fName, lName, COUNT(*)
  FROM Staff s, PropertyForRent p
  WHERE s.staffNo = p.staffNo
  GROUP BY s.staffNo, fName, lName)
  UNION
  (SELECT staffNo, fName, lName, 0
  FROM Staff
  WHERE staffNo NOT IN
  (SELECT DISTINCT staffNo
     FROM PropertyForRent));
```

***UPDATE***

```
UPDATE TableName
SET columnName1 = dataValue1
  [, columnName2 = dataValue2...]
[WHERE searchCondition]
```

•TableName can be name of a base table or an updatable view.
•SET clause specifies names of one or more columns that are to be updated.
•WHERE clause is optional:
•if omitted, named columns are updated for all rows in table;
•if specified, only those rows that satisfy searchCondition are updated.
•New dataValue(s) must be compatible with data type for corresponding column.
Give all staff a 3% pay increase.

```
      UPDATE Staff
   SET salary = salary*1.03;
```

Give all Managers a 5% pay increase.

```
  UPDATE Staff
  SET salary = salary*1.05
  WHERE position = ‘Manager’;
```

Promote David Ford (staffNo=‘SG14’) to Manager and change his salary to £18,000.

```
  UPDATE Staff
  SET position = ‘Manager’, salary = 18000
  WHERE staffNo = ‘SG14’;
```

***DELETE***

```
DELETE FROM TableName
[WHERE searchCondition]
```

•TableName can be name of a base table or an updatable view.
searchCondition is optional; if omitted, all rows are deleted from table. This does not delete table. If search_condition is specified, only those rows that satisfy condition are deleted.
Delete all viewings that relate to property PG4.

```
  DELETE FROM Viewing
  WHERE propertyNo = ‘PG4’;
```

Delete all records from the Viewing table.

```
  DELETE FROM Viewing;
```