

Open Database -> go to the db location -> open file (.db file)

The screenshot shows a database management application window. At the top, there are tabs for 'New Database', 'Open Database' (which is selected), 'Write Changes', 'Recent Changes', 'Open Project', 'Save Project', 'Attach Database', and 'Close Database'. Below the tabs is a toolbar with icons for 'Create Table', 'Create Index', 'Print', 'Edit Expressions', and 'Execute SQL'.

The main area has two panes. The left pane displays the database structure with a tree view:

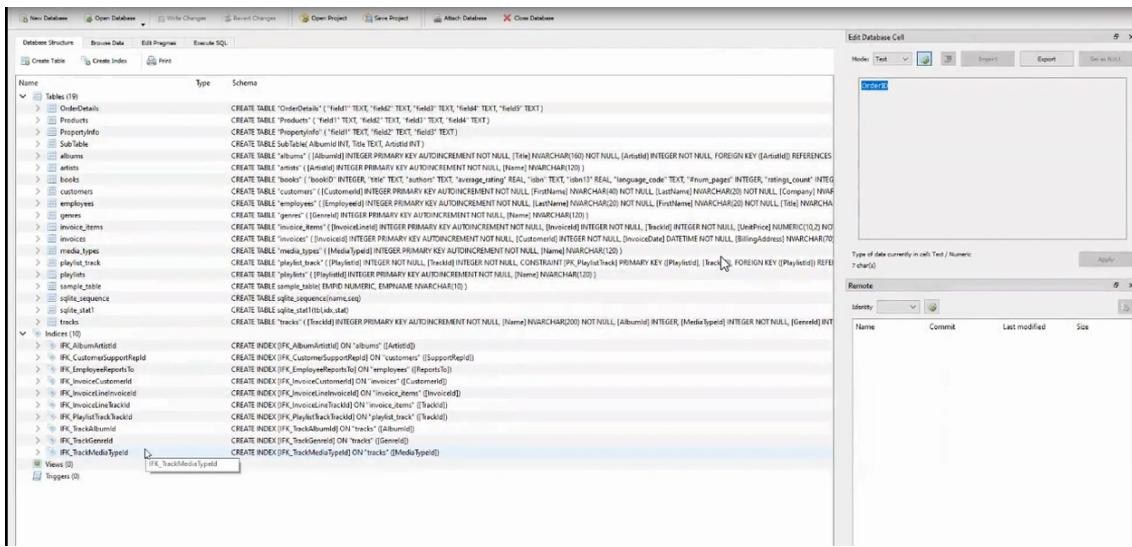
- Tables (19):**
 - OrderDetails
 - Order
 - Product
 - ProductInfo
 - Region
 - SubTable
 - albums
 - artists
 - books
 - customers
 - employees
 - genres
 - invocation_items
 - invokes
 - media_type
 - playlist_track
 - playlists
 - sample_table
 - sqlite_sequence
 - sqlite_stat
 - tracks
- Indices (10):**
 - FK_AlbumArtistId
 - FK_CustomerSupportrepid
 - FK_EmployeeReportsTo
 - FK_InvoiceCustomerid
 - FK_InvoiceLineInvoiceid
 - FK_InvoiceLineTrackid
 - FK_PlaylistTrackid
 - FK_TrackGenreid
 - FK_TrackMediaTypid
- Views (0)**
- Triggers (0)**

The right pane contains three windows:

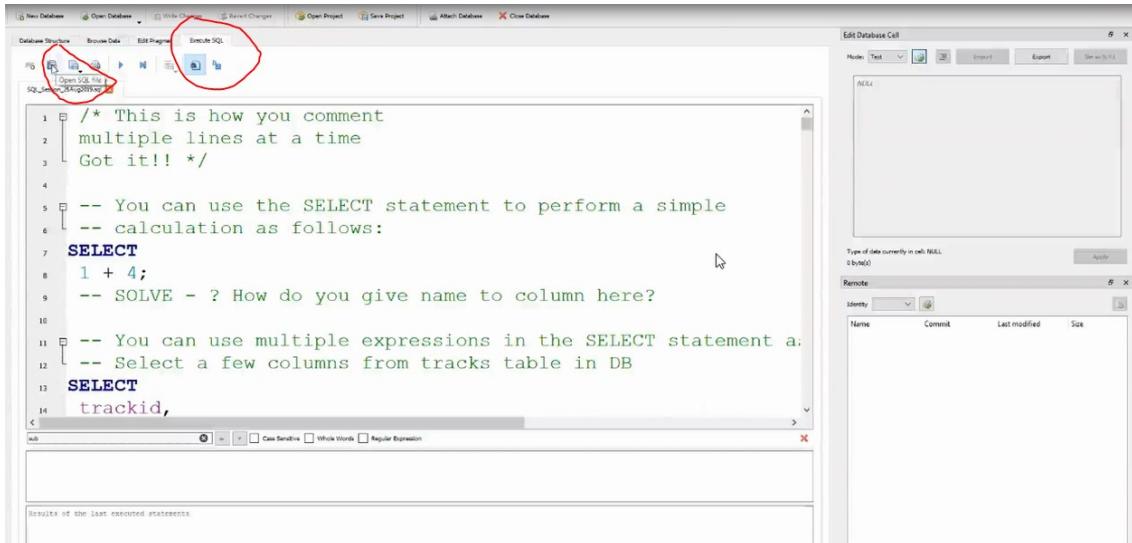
- Edit Database Cell:** Shows a table structure with columns: OrderId, OrderDate, EmployeeId, CustomerId, and ShipAddress. A tooltip indicates the current cell value is 'Order'.
- Remote:** A small window showing connection details.
- Identity:** A window listing columns with their data types, commit status, last modified time, and size.

Below the main interface is a file explorer window titled 'Choose a database file' showing the local file system. It lists a single file: 'sample_db1.db' located at 'This PC > Documents > Jigsaw Courses > Courses > SQL - Codes and Datasets'. The file was modified on 13-10-2019 at 04:30 PM and is 2,616 KB in size.

The bottom of the screen shows the operating system's taskbar with several pinned icons.



Execute SQL -> open SQL file -> go to location -> open file (.sql file)



Database structure tab -> all tables are listed over here

Execute SQL -> code is written over here

Simple math calculation

The screenshot shows a SQL query window with the following code:

```

1 /* This is how you comment
2 multiple lines at a time
3 Got it!! */

4
5 -- You can use the SELECT statement to perform a simple
6 -- calculation as follows:
7 SELECT
8     1 + 4;
9 -- SOLVE - ? How do you give name to column here?

10
11 -- You can use multiple expressions in the SELECT statement as
12 -- Select a few columns from tracks table in DB
13 SELECT
14     trackid,
15     name,
16     composer,
17     unitprice

```

The results pane shows the output of the query:

```

1 + 4
1 5

Results: 1 rows returned in 1ms
Ex. line 7:
SELECT
1 + 4;

```

Columnname is available with this piece of code

The screenshot shows a SQL query window with the following code:

```

1 /* This is how you comment
2 multiple lines at a time
3 Got it!! */

4
5 -- You can use the SELECT statement to perform a simple
6 -- calculation as follows:
7 SELECT
8     1 + 4 as columnName;
9 -- SOLVE - ? How do you give name to column here?

10
11 -- You can use multiple expressions in the SELECT statement as
12 -- Select a few columns from tracks table in DB
13 SELECT
14     trackid,
15     name,
16     composer,
17     unitprice

```

The results pane shows the output of the query:

```

columnName
1 5

Results: 1 rows returned in 1ms
Ex. line 7:
SELECT
1 + 4 as columnName;

```

Trackid table

The screenshot shows the Object Explorer with the 'Tables' node expanded. A red box highlights the definition of the 'Trackid' table.

Table Definition:

```

CREATE TABLE [dbo].[Trackid] (
    [trackid] INT NOT NULL,
    [Name] NVARCHAR(100) NOT NULL,
    [Artist] NVARCHAR(100),
    [Composer] NVARCHAR(100),
    [Milliseconds] INT NOT NULL,
    [Bytes] INT NOT NULL,
    [UnitPrice] NUMERIC(10, 2) NOT NULL
)

```

Indexes:

- PK_Trackid (Primary Key)
- IX_CustOrderDetailId (Nonclustered)
- IX_EmployeeReportsTo (Nonclustered)
- IX_InvLineInvoiceId (Nonclustered)
- IX_InvLineTrackId (Nonclustered)
- IX_PayLoadId (Nonclustered)
- IX_SalesDetailId (Nonclustered)

Browse data tab -> to have a look at the particular table

The screenshot shows the 'Browse Data' tab of a database interface. On the left, there's a tree view with 'Database Structure' expanded, showing 'Tables' and 'Index'. The main area displays the 'tracks' table with columns: TrackId, Name, AlbumId, MediaTypeId, GenreId, Composer, Milliseconds, Bytes, and UnitPrice. There are 24 rows of data. A context menu is open over the 10th row (Evil Walks). The menu items are: 'References generated by this row' (disabled), 'Hold Ctrl+Shift and click to jump there', 'New...', 'Edit...', 'Delete Record...', 'Import...', 'Export...', and 'Set as NULL...'. To the right of the table is an 'Edit Database Cell' window for the 10th row, showing the value '1' in the 'Value' field. The window has tabs for 'Mode' (set to 'Text'), 'Import', 'Export', and 'Set as NULL'. Below the table is a 'Remote' section with a table for 'Identity'.

Data can be filtered over here

This screenshot shows the same 'Browse Data' tab and 'tracks' table setup as the first one. A context menu is open over the 10th row (Evil Walks). The menu items are: 'References albums/albumsId' (disabled), 'Hold Ctrl+Shift and click to jump there', 'New...', 'Edit...', 'Delete Record...', 'Import...', 'Export...', and 'Set as NULL...'. The 'Edit Database Cell' window is also present, showing the value '1' in the 'Value' field. The 'Remote' section below the table is empty.

Database Structure Browse Data Edit Programs Execute SQL

Table: tracks

TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
1	For Those About To Rock (We Salute You)	1	1	1	Angus Young, Malcolm Young, Brian Johnson	343719	11170334	0.99
2	43	6	1	1	Alain Monsette ...	300355	9753256	0.99
3	86	9	1	3	Apocalyptica	329252	10422447	0.99
4	134	14	1	3	REOZ	235833	7726948	0.99
5	408	35	1	3	Moloney/Morris/R...	155428	5075048	0.99
6	427	Who Wants To Live...	36	1	Brian May	297691	9577577	0.99
7	446	I Was Made For Liv...	37	1	Paul Stanley, Vinc...	271360	9108678	0.99
8	495	Cry For Love	40	1	Boss/David Cover...	293015	9597075	0.99
9	526	Fera Da Ordem	23	1	REOZ	354011	11746781	0.99
10	652	Fernando	52	1	11	127482	4506873	0.99
11	683	Fortunate Son	54	1	J.C. Fogerty	140329	4617559	0.99
12	700	Wrote A Song For Yo...	55	1	J.C. Fogerty	296385	9675875	0.99
13	706	Before You Accuse...	55	1	J.C. Fogerty	207804	6815126	0.99
14	713	Lookin' For A Reason...	55	1	J.C. Fogerty	209789	6933113	0.99
15	782	Never Before	62	1	Ian Gillan/Ian Pric...	239830	7832790	0.99
16	788	Soon Forgotten	63	1	Ian Gillan, Roger ...	287791	9401383	0.99
17	804	Fortuneteller	64	1	Blackmore, Glorv...	349315	11369671	0.99
18	815	Soldier Of Fortune	65	1	D.Coverdale/R.E.B...	193750	6315321	0.99
19	910	Before You Accuse...	73	1	Eugene McDaniel	224239	7456807	0.99
20	954	Cuckoo For Cuckoo	76	1	Mike Bordin, Billy ...	222992	7388369	0.99
21	959	King For A Day	76	1	Mike Bordin, Billy ...	395859	13163733	0.99
22	980	Fern De Tóquio	78	1	REOZ	169273	5588756	0.99
23	988	Desafiero	78	1	REOZ	174524	5835861	0.99
24	1041	For Once In My Life	83	1	Orlando murden/Y...	171154	5557537	0.99

Database Structure Browse Data Edit Programs Execute SQL

Table: tracks

TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
101	Be Yourself	11	1	4	Correll, Commer...	279484	9106160	0.99
102	Doesn't Remind Me	11	1	4	Correll, Commer...	255869	8357387	0.99
103	Drawn Me Slowly	11	1	4	Correll, Commer...	233691	7669178	0.99
104	Heaven's Dead	11	1	4	Correll, Commer...	276688	9006158	0.99
105	The Warm	11	1	4	Correll, Commer...	237714	7710800	0.99
106	Man Or Animal	11	1	4	Correll, Commer...	231915	7542942	0.99
107	Yesterday To Tomo...	11	1	4	Correll, Commer...	273763	8944205	0.99
108	Dandelion	11	1	4	Correll, Commer...	279125	9003592	0.99
109	#1 Zero	11	1	4	Correll, Commer...	299102	9731988	0.99
110	The Curse	11	1	4	Correll, Commer...	309786	10029406	0.99
111	Money	12	1	5	Berry Gordy, Jr./Ja...	147991	2265897	0.99
112	Long Tall Sally	12	1	5	Endris Johnson/Li...	106396	1707894	0.99
113	Bad Boy	12	1	5	Larry Williams	116688	1862126	0.99
114	Twist And Shout	12	1	5	Bert Russell/Phil M...	161123	2582553	0.99
115	Please Mr. Postman	12	1	5	Brins Holland/Fred...	137639	2206986	0.99
116	C'Mon Everybody	12	1	5	Eddie Cochran/Har...	140199	2247946	0.99
117	Rock 'N' Roll Music	12	1	5	Chuck Berry	141923	2276788	0.99
118	Slow Down	12	1	5	Larry Williams	162265	2616981	0.99
119	Roadrunner	12	1	5	Bo Diddley	143995	2301989	0.99
120	Carol	12	1	5	Chuck Berry	143830	2306619	0.99
121	Good Golly Miss M...	12	1	5	Little Richard	106266	1704918	0.99
122	20 Flight Rock	12	1	5	Ned Flanders	107807	1299960	0.99
123	Quidnunc	13	1	2	Billy Colham	261851	8538199	0.99
124	Snoopy's search...R...	13	1	2	Billy Colham	456971	15075616	0.99

Selecting few columns from the table (select * to select all the columns)

Database Structure Browse Data Edit Programs Execute SQL

SQL_Demo_15Aug2014

```

SELECT
  1 + 4;
-- SOLVE - ? How do you give name to column here?

-- You can use multiple expressions in the SELECT statement as
-- Select a few columns from tracks table in DB
SELECT
  trackid,
  name,
  composer,
  unitprice
FROM
  tracks;

/* SOLVE - ? How to select All the columns from tracks table?
-- Way/s - Which one is easier and which is recommended? */

```

TrackId	Name	Composer	UnitPrice
1	For Those About To Rock (We Salute You)	Angus Young, Malcolm Young, Brian Johnson	0.99
2	Balls to the Wall	REOZ	0.99

Results: 3503 rows returned in 11ms
 1st line 12:
 -- Select a few columns from tracks table in DB
 SELECT
 trackid,

Addition of two columns

The screenshot shows a SQL Server Management Studio (SSMS) interface. In the top-left window, there is a query editor with the following code:

```
7  SELECT
8      1 + 4;
9      -- SOLVE - ? How do you give name to column here?
10
11     -- You can use multiple expressions in the SELECT statement as
12     -- Select a few columns from tracks table in DB
13     SELECT
14         trackid,
15         name,
16         composer,
17         unitprice,
18        Milliseconds + Bytes as name
19     FROM
20         tracks;
21
22     /* SOLVE- ? How to select All the columns from tracks table?
23     - Way/s - Which one is easier and which is recommended? */
24
25
26
27
28
29
30
31
```

The results grid below the query editor displays two rows of data:

TrackId	Name	Composer	UnitPrice
1	For Those About To Rock (We Salute You)	Angus Young, Malcolm Young, Brian Johnson	0.99
2	Bells to the Well		0.99

At the bottom of the results grid, it says "Result: 3503 rows returned in 1ms".

This code will select the all the rows and there will be duplicate city names present in the result

(59 rows)

The screenshot shows a SQL Server Management Studio (SSMS) interface. In the top-left window, there is a query editor with the following code:

```
16     composer,
17     unitprice
18   FROM
19     tracks;
20
21     /* SOLVE- ? How to select All the columns from tracks table?
22     - Way/s - Which one is easier and which is recommended? */
23
24     -- Get the cities from where customer belongs
25
26     SELECT
27         city
28     FROM
29         customers;
30     -- SOLVE - ? What is the issue with above query?
31
```

The results grid below the query editor displays two rows of data:

City
São José dos Campos
Stuttgart

At the bottom of the results grid, it says "Result: 59 rows returned in 1ms".

Distinct will select only unique cities (53 rows)

The screenshot shows the SQL Server Management Studio interface. On the left, a query window displays the following T-SQL code:

```
28 FROM
29   customers;
30 -- SOLVE - ? What is the issue with above query?
31
32
33 -- Use of DISTINCT
34 SELECT DISTINCT
35   city
36   FROM
37   customers;
38
39 -- SOLVE - ? Get the companies of the customers
40 -- What is unusual about result?
41
42 SELECT
43   company
44   FROM
```

The results grid below shows the output:

City
São José dos Campos
Stuttgart

Details at the bottom of the results grid:

Result: 53 rows returned in 1ms
At line 34:
SELECT DISTINCT
 city
 FROM

Orderby will sort the data

The screenshot shows the SQL Server Management Studio interface. On the left, a query window displays the following T-SQL code:

```
56 FROM
57   tracks;
58
59 -- Now get the same data sorted based on AlbumId column in ascending order
60
61 SELECT
62   name,
63   milliseconds,
64   albumid
65   FROM
66   tracks
67 ORDER BY
68   albumid ASC;
69
70
71 -- Now sort the sorted result (by AlbumId) above...
72 -- by theMilliseconds column in descending order
73
74 SELECT
```

The results grid below shows the output:

City
São José dos Campos
Stuttgart

Details at the bottom of the results grid:

Result: 53 rows returned in 1ms
At line 34:
SELECT DISTINCT
 city
 FROM

ASC : Ascending ; DESC : Descending

The screenshot shows the SQL Server Management Studio interface. On the left, a query window displays the following T-SQL code:

```
67 tracks
68 ORDER BY
69   albumid ASC;
70
71 -- Now sort the sorted result (by AlbumId) above...
72 -- by theMilliseconds column in descending order
73
74 SELECT
75   name,
76   milliseconds,
77   albumid
78   FROM
79   tracks
80 ORDER BY
81   albumid ASC,
82   milliseconds DESC;
```

The results grid below shows the output:

City
São José dos Campos
Stuttgart

Details at the bottom of the results grid:

Result: 53 rows returned in 1ms
At line 34:
SELECT DISTINCT
 city
 FROM

Using column positions to sort

The screenshot shows the SQL Server Management Studio interface. In the top-left window, there is a query editor with the following SQL code:

```

82     milliseconds DESC;
83
84 -- Using column positions in ORDER BY
85 SELECT
86     name,
87     milliseconds,
88     albumid
89     FROM
90     tracks
91     ORDER BY
92         3,2;
93
94 -- Use of LIMIT clause
95 -- get the first 10 rows in the tracks table
96 SELECT
97     trackId,
98     name
99     FROM
100    tracks
101   ORDER BY
102      1
103
104 -- get 10 rows starting from the 10th row in the tracks table
105 SELECT
106     trackId,
107     name
108     FROM
109     tracks
110   LIMIT 10 OFFSET 10;

```

The results pane below shows the output of the last two queries:

City
1 São José dos Campos
2 Stuttgart

Below the results pane, the status bar indicates: "Results: 33 rows returned in 1ms At line 14: SELECT DISTINCT city FROM".

Limit will give only given no of rows of data, offset will skip given no of rows

The screenshot shows the SQL Server Management Studio interface. In the top-left window, there is a query editor with the following SQL code:

```

94 -- Use of LIMIT clause
95 -- get the first 10 rows in the tracks table
96 SELECT
97     trackId,
98     name
99     FROM
100    tracks
101   LIMIT 10;
102
103 -- get 10 rows starting from the 10th row in the tracks table
104 SELECT
105     trackId,
106     name
107     FROM
108     tracks
109   LIMIT 10 OFFSET 10;

```

The results pane below shows the output of the last two queries:

City
1 São José dos Campos
2 Stuttgart

Below the results pane, the status bar indicates: "Results: 33 rows returned in 1ms At line 34: SELECT DISTINCT city FROM".

sort and limit on single piece of code

The screenshot shows the SQL Server Management Studio interface. In the top-left window, there is a query editor with the following SQL code:

```

112
113
114
115
116 SELECT
117     trackid,
118     name,
119     bytes
120     FROM
121     tracks
122     ORDER BY
123     bytes DESC
124   LIMIT 10;
125
126 -- SOLVE NUGGET - ? Get all customers and their first name who
127 -- select customerid, firstname, company
128     from customers

```

The results pane below shows the output of the last two queries:

TrackId	Name
11	C.O.D.
12	Breaking The Rules

Below the results pane, the status bar indicates: "Results: 10 rows returned in 1ms At line 10: -- select customerid, firstname, company from customers".

Select data where company is null (company name is absent)

The screenshot shows a SQL Server Management Studio interface. In the top-left window, there is a query window with the following code:

```
118 name,
119 bytes
120 FROM
121 tracks
122 ORDER BY
123 bytes DESC
124 LIMIT 10;

125
126 -- SOLVE NUGGET - ? Get all customers and their first name who
127 -- select customerid, firstname, company
128 --from customers
129 --where company is NULL;

130
131 --Use of where clause
132 -- Get all tracks in the album id 1
133
```

Below the code is a results grid:

TrackId	Name
1	C.O.D.
2	Breaking The Rules

At the bottom of the results grid, it says "Result: 10 rows returned in 1ms".

In the top-right corner, there is a properties window for a table named "tracks". It shows the column "bytes" with a type of "tinyint" and a value of "0". A tooltip says "Type of data currently in cell: NULL".

Where statement is used in this piece of code

The screenshot shows a SQL Server Management Studio interface. In the top-left window, there is a query window with the following code:

```
118 name,
119 bytes
120 FROM
121 tracks
122 ORDER BY
123 bytes DESC
124 LIMIT 10;

125
126 -- SOLVE NUGGET - ? Get all customers and their first name who
127 select customerid, firstname, company
128 from customers
129 where company is NULL;

130
131 --Use of where clause
132 -- Get all tracks in the album id 1
133
```

Below the code is a results grid:

CustomerID	FirstName	Company
48	Mario	NULL
49	Puja	NULL

At the bottom of the results grid, it says "Result: 49 rows returned in 1ms".

In the top-right corner, there is a properties window for a table named "customers". It shows the column "company" with a type of "nvarchar(60)" and a value of "NULL". A tooltip says "Type of data currently in cell: NULL".

The screenshot shows a SQL Server Management Studio interface. In the top-left window, there is a query window with the following code:

```
130
131 --Use of where clause
132 -- Get all tracks in the album id 1
133
134 SELECT
135 name,
136 milliseconds,
137 bytes,
138 albumid
139 FROM
140 tracks
141 WHERE
142 albumid = 1;
143
144 -- SOLVE: Get tracks on the album 1 that have the length greater
145
```

Below the code is a results grid:

CustomerID	FirstName	Company
35	Ivanov	NULL
36	Hugh	NULL

At the bottom of the results grid, it says "Result: 49 rows returned in 1ms".

In the top-right corner, there is a properties window for a table named "customers". It shows the column "company" with a type of "nvarchar(60)" and a value of "NULL". A tooltip says "Type of data currently in cell: NULL".

And will apply to conditions inside where statement

The screenshot shows a SQL query in the Query Editor window:

```
145  
146  
147  
148 SELECT  
149 name,  
150 milliseconds,  
151 bytes,  
152 albumid  
153 FROM  
154 tracks  
155 WHERE  
156 albumid = 1  
157 AND milliseconds > 250000;  
158  
159  
160 --- Find which tracks are composed by all people with 'Smith' in  
161 wild card characters.
```

The results pane shows two rows of data:

	Name	Milliseconds	Bytes	AlbumId
1	For Those About To Rock (We Salute You)	343719	11170334	1
2	Evil Walks	263497	8611245	1

Details pane: Type of data currently in cell: NULL

The screenshot shows a similar SQL query in the Query Editor window:

```
145  
146  
147  
148 SELECT  
149 name,  
150 milliseconds,  
151 bytes,  
152 albumid  
153 FROM  
154 tracks  
155 WHERE  
156 albumid = 1  
157 AND milliseconds >= 250000;
```

The results pane shows the same two rows of data:

	Name	Milliseconds	Bytes	AlbumId
1	For Those About To Rock (We Salute You)	343719	11170334	1
2	Evil Walks	263497	8611245	1

Details pane: Type of data currently in cell: NULL

In some of the query languages words can be used instead of signs

The screenshot shows the same SQL query in the Query Editor window, with a red circle highlighting the condition in the WHERE clause:

```
145  
146  
147  
148 SELECT  
149 name,  
150 milliseconds,  
151 bytes,  
152 albumid  
153 FROM  
154 tracks  
155 WHERE  
156 albumid = 1  
157 AND milliseconds > 250000;  
158  
159 gt - >, ge >= , lt - <, le <=, =
```

The results pane shows the same two rows of data:

	Name	Milliseconds	Bytes	AlbumId
1	For Those About To Rock (We Salute You)	343719	11170334	1
2	Evil Walks	263497	8611245	1

Details pane: Type of data currently in cell: NULL

To select data rows which contains part of the string (% signs can be used)

% before word – another words/letters are present before this word

% after word – another words/letters are present after this word

% on both side – another words/letters are present before & after this word

SQL Server Management Studio screenshot showing a query window with the following code:

```
-- Find which tracks are composed by all people with 'Smith' in
-- Wildcard characters
SELECT
    name,
    albumid,
    composer
FROM
    tracks
WHERE
    composer LIKE '%Smith%'
ORDER BY
    albumid;
```

The results grid shows two rows:

Name	AlbumId	Composer
Killing Floor	19	Adrian Smith
Machete Killin	16	Adrian Smith

Message bar at the bottom: Result: 2 rows returned in 0ms
At line 1, col 1:
SELECT
 name,
 albumid,

SQL Server Management Studio screenshot showing a query window with the following code:

```
-- Use of wildcards % and _
SELECT
    trackid,
    name
FROM
    tracks
WHERE
    name LIKE 'Wild%';
```

The results grid shows two rows:

TrackId	Name
1245	Wildest Dreams
1973	Wild Side

Message bar at the bottom: Result: 2 rows returned in 0ms
At line 1, col 1:
SELECT
 trackid,
 name

SQL Server Management Studio screenshot showing a query window with the following code:

```
SELECT
    trackid,
    name
FROM
    tracks
WHERE
    name LIKE '%Wild';
```

The results grid shows two rows:

TrackId	Name
1245	Wildest Dreams
1973	Wild Side

Message bar at the bottom: Result: 2 rows returned in 0ms
At line 1, col 1:
SELECT
 trackid,
 name

Or condition inside where

The screenshot shows a SQL query window with the following code:

```

199
200    -- SOLVE: Find tracks that have media type id 2 or 3
201
202
203    SELECT
204        name,
205        albumid,
206        mediatypeid
207    FROM
208        tracks
209    WHERE
210        mediatypeid = 2 or mediatypeid = 3;
211
212    -- SOLVE: What if I have to chose data for more than 25 different
213
214

```

Below the code is a results grid showing two rows of data:

	Name	AlbumId	MediaTypeId
1	Balls to the Wall	2	2
2	Fat As a Shark	3	2

In condition – alternate for or condition (if applied on single column)

The screenshot shows a SQL query window with the following code:

```

214
215
216    --Use of IN operator and Not In operator
217    SELECT
218        name, albumid, mediatypeid
219    FROM
220        tracks
221    WHERE
222        mediatypeid IN (2, 3);
223
224    -- list of tracks whose genre id is not in a list of (1,2,3)
225    SELECT
226        trackid,
227        name,
228        genreid
229    FROM
230        tracks
231

```

Below the code is a results grid showing two rows of data:

	Name	AlbumId	MediaTypeId
1	Balls to the Wall	2	2
2	Fat As a Shark	3	2

Notin condition – opposite of in

The screenshot shows a SQL query window with the following code:

```

220
221    tracks
222    WHERE
223        mediatypeid NOT IN (2, 3);
224
225    -- list of tracks whose genre id is not in a list of (1,2,3)
226    SELECT
227        trackid,
228        name,
229        genreid
230    FROM
231        tracks
232    WHERE
233        genreid NOT IN (1,2,3);
234
235    -- Use of BETWEEN operator
236    -- finds invoices whose total is between 14.01 and 10.06.

```

Below the code is a results grid showing two rows of data:

	Name	AlbumId	MediaTypeId
1	Balls to the Wall	2	2
2	Fat As a Shark	3	2

Between condition

The screenshot shows a SQL Server Management Studio window with a query editor and a results grid. The query uses the BETWEEN operator to find invoices between two total amounts:

```
232 genreid NOT IN (1,2,3);
233
234
235 -- Use of BETWEEN operator
236 -- finds invoices whose total is between 14.91 and 18.86:
237
238     SELECT
239         InvoiceId,
240         BillingAddress,
241         Total
242     FROM
243         invoices
244     WHERE
245         Total BETWEEN 14.91 AND 18.86
246     ORDER BY
247         Total;
```

The results grid shows two rows of data:

Name	AlbumId	MediaTypeId
Balls to the Wall	2	2
Fat As a Shark	3	2

notbetween condition

The screenshot shows a SQL Server Management Studio window with a query editor and a results grid. The query uses the NOT BETWEEN operator to exclude invoices between two total amounts:

```
250
251
252
253
254
255
256
257
258
259     SELECT
260         InvoiceId,
261         BillingAddress,
262         Total
263     FROM
264         invoices
265     WHERE
266         Total NOT BETWEEN 1 AND 20
267     ORDER BY
268         Total;
```

Between condition on dates

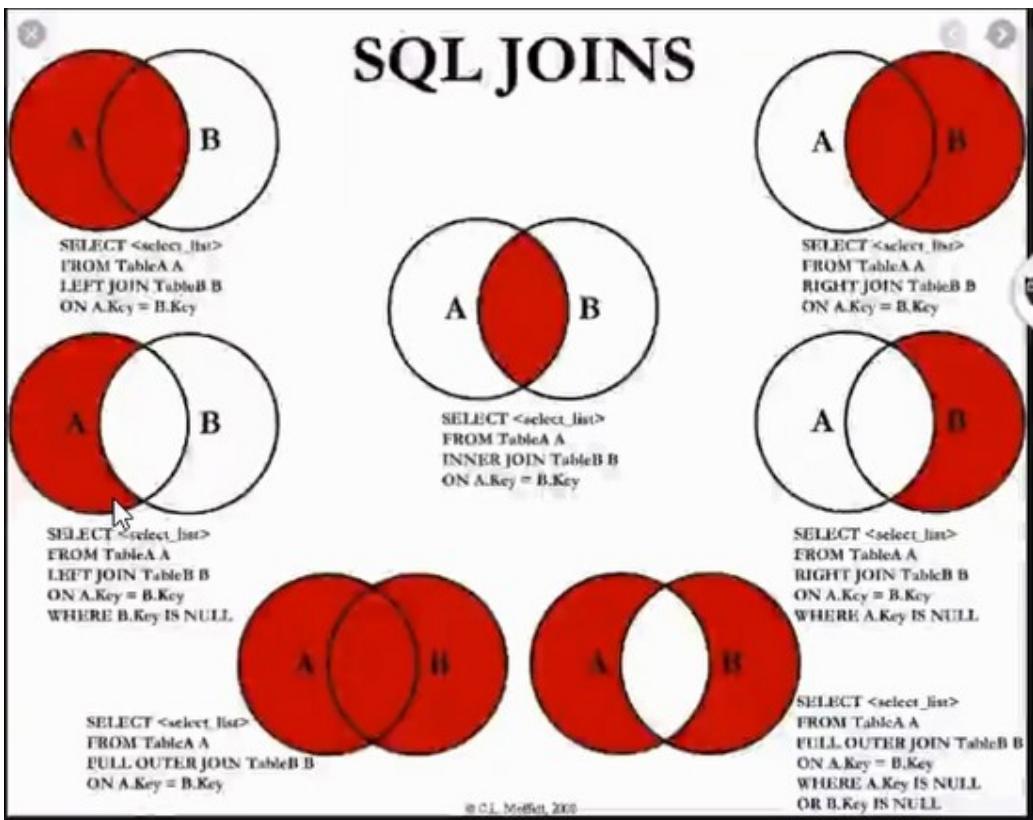
The screenshot shows a SQL Server Management Studio window with a query editor and a results grid. The query uses the BETWEEN operator on the InvoiceDate column to find invoices from January 1, 2010, to January 31, 2010:

```
265
266
267 -- finds invoices whose invoice dates are from January 1 2010 .
268
269     SELECT
270         InvoiceId,
271         BillingAddress,
272         InvoiceDate,
273         Total
274     FROM
275         invoices
276     WHERE
277         InvoiceDate BETWEEN '2010-01-01' AND '2010-01-31'
278     ORDER BY
279         InvoiceDate;
```

The results grid shows two rows of data:

Name	AlbumId	MediaTypeId
Balls to the Wall	2	2
Fat As a Shark	3	2

SQL Joins



Inner join on tracks and album

Trackid and name from tracks table

Title from album table

The screenshot shows a SQL query window with the following code:

```
280
281 -- JOINS
282 [-- Inner join
283
284 -- get the album titles for all tracks
285 SELECT
286     trackid,
287     name,
288     title
289 FROM
290     tracks
291 INNER JOIN
292     albums ON albums.albumid = tracks.albumid;
```

Below the code, a message says "How do you validate join is correct?". The results pane displays a table with three columns: TrackId, Name, and Title. The data is as follows:

TrackId	Name	Title
1	For Those About To Rock (We Salute You)	For Those About To Rock We Salute You
2	Put The Finger On You	For Those About To Rock We Salute You

Result: 2 rows returned in 0ms
At line 255:
SELECT
 trackid,
 name,

Usage of alias for table names

The screenshot shows a SQL query window with the following code:

```
291 INNER JOIN
292     albums ON albums.albumid = tracks.albumid;
293
294 SELECT
295     a.trackid,
296     a.name,
297     b.title
298 FROM
299     tracks a
300 INNER JOIN
301     albums b ON b.albumid = a.albumid;
```

Below the code, a message says "How do you validate join is correct?". The results pane displays a table with three columns: TrackId, Name, and Title. The data is as follows:

TrackId	Name	Title
1	For Those About To Rock (We Salute You)	For Those About To Rock We Salute You
2	Put The Finger On You	For Those About To Rock We Salute You

Result: 2 rows returned in 0ms
At line 254:
SELECT
 trackid,
 name,

Getting albumids as well from both tables (to validate the join)

```

303
304    -- How do you validate join is correct?
305    SELECT
306        trackid,
307        name,
308        tracks.albumid AS album_id_tracks,
309        albums.albumid AS album_id_albums,
310        title
311    FROM
312        tracks
313    INNER JOIN albums ON albums.albumid = tracks.albumid;
314
315
316
317    -- SOLVE - ? How to get tracks and albums of artist with ID = 10
318

```

The screenshot shows the results of the query in a table:

TrackId	Name	album_id_tracks	album_id_albums	Title
1	For Those About To Rock (We Salute You)	1	1	For Those About To Rock We Salute You
2	Put The Finger On You	1	1	For Those About To Rock We Salute You

Results: 2 rows returned in 1 items
At line 316:
SELECT
 trackid,
 name,

Joining multiple tables

```

316
317    -- SOLVE - ? How to get tracks and albums of artist with ID = 10
318

```



```

321
322    SELECT
323        trackid,
324        tracks.name AS Track,
325        albums.title AS Album,
326        artists.name AS Artist
327    FROM
328        tracks
329    INNER JOIN albums ON albums.albumid = tracks.albumid
330    INNER JOIN artists ON artists.artistid = albums.artistid
331    WHERE
332        artists.artistid = 10;
333
334
335    -- SOLVE: Find the artists who do not have any albums
336

```

The screenshot shows the results of the first query in a table:

TrackId	Name	album_id_tracks	album_id_albums	Title
1	For Those About To Rock (We Salute You)	1	1	For Those About To Rock We Salute You
2	Put The Finger On You	1	1	For Those About To Rock We Salute You

Results: 2 rows returned in 1 items
At line 321:
SELECT
 trackid,
 name,


```

321
322    SELECT
323        trackid,
324        tracks.name AS Track,
325        albums.title AS Album,
326        artists.name AS Artist
327

```

The screenshot shows the results of the second query in a table:

TrackId	Track	Album	Artist
1	Quadrant	The Best Of Billy Cobham	Billy Cobham
2	Snoopy's search-Red Baron	The Best Of Billy Cobham	Billy Cobham
3	Spanish moss~A sound portrait~Spanish moss	The Best Of Billy Cobham	Billy Cobham
4	Moon germs	The Best Of Billy Cobham	Billy Cobham
5	Saturn	The Best Of Billy Cobham	Billy Cobham
6	The pleasant pheasant	The Best Of Billy Cobham	Billy Cobham
7	Solo-Panhandler	The Best Of Billy Cobham	Billy Cobham
8	Do what cha wanna	The Best Of Billy Cobham	Billy Cobham

Results: 8 rows returned in 10ms
At line 321:
SELECT
 trackid,
 tracks.name AS Track,
 albums.title AS Album,
 artists.name AS Artist
FROM
 tracks
INNER JOIN albums ON albums.albumid = tracks.albumid
INNER JOIN artists ON artists.artistid = albums.artistid
WHERE
 artists.artistid = 10;

```

334
335 -- SOLVE: Find the artists who do not have any albums
336

```

The screenshot shows the SQLite Studio interface. On the left is the SQL editor window containing the following code:

```

333
334 -- SOLVE: Find the artists who do not have any albums
335
336
337
338 SELECT
339   artists.ArtistId,
340   albumId
341   FROM
342   artists
343 LEFT JOIN albums ON albums.artistid = artists.artistid
344 ORDER BY
345   albumid;
346
347 --OR
348 SELECT
349   artists.ArtistId

```

Below the code is the results pane, which displays a table with two rows:

ArtistId	AlbumId
53	176
54	177

The status bar at the bottom indicates "Result: 422 rows returned in 2ms".

Alternate way

```

347 --OR
348 SELECT
349   artists.ArtistId,
350   albumId
351   FROM
352   artists
353 LEFT JOIN albums ON albums.artistid = artists.artistid
354 WHERE
355   albumid IS NULL;
356
357
358 -- Note - SQLite does not support RIGHT JOIN and FULL OUTER jo
359

```

Groupby (like pivot table in excel)

The screenshot shows the SQLite Studio interface. On the left is the SQL editor window containing the following code:

```

357
358 -- Note - SQLite does not support RIGHT JOIN and FULL OUTER jo
359
360 -- GROUP BY
361 -- Find the number of tracks per album
362
363 SELECT
364   albumid,
365   COUNT(trackid) AS track_count
366   FROM
367   tracks
368 GROUP BY
369   albumid;
370
371 -- order above result by count of tracks
372 SELECT
373   albumid

```

Below the code is the results pane, which displays a table with two rows:

AlbumId	track_count
1	10
2	1

The status bar at the bottom indicates "Result: 347 rows returned in 3ms".

Sorting on top of groupby

The screenshot shows a SQL query window with the following code:

```

366 FROM
367 tracks
368 GROUP BY
369 albumid;
370
371 -- order above result by count of tracks
372 SELECT
373 albumid,
374 COUNT(trackid) as track_count
375 FROM
376 tracks
377 GROUP BY
378 albumid
379 ORDER BY 2 DESC;
380
381 -- Get the number of tracks per album along with album title

```

Below the code is a results grid:

AlbumId	track_count
1	57
2	34

And a status bar at the bottom:

Results: 147 rows returned in 20ms
At line 372:
SELECT
albums,
COUNT(trackid) as track_count

The screenshot shows a SQL query window with the following code:

```

375 FROM
376 tracks
377 GROUP BY
378 albumid
379 ORDER BY 2 DESC;
380
381 -- Get the number of tracks per album along with album title
382
383 select a.albumID , b.title, count (trackid) as track_count
384 from
385 tracks as a
386 left join
387 albums as b on a.albumid = b.albumid
388 group by 1,2;
389
390 -- GROUP BY with HAVING clause - to filter by aggregate value
391 -- get the albums that have more than 15 tracks

```

Below the code is a results grid:

AlbumId	Title	track_count
1	For Those About To Rock We Salute You	10
2	Bells to the Well	1

And a status bar at the bottom:

Results: 147 rows returned in 40ms
At line 391:
select a.albumID , b.title, count (trackid) as track_count
from
tracks as a

Having condition – to be applied on top of groupby to filter aggregated data

The screenshot shows a SQL query window with the following code:

```

390 -- GROUP BY with HAVING clause - to filter by aggregate value
391 -- get the albums that have more than 15 tracks
392
393 SELECT
394 tracks.albumid,
395 title,
396 COUNT(trackid)
397 FROM
398 tracks
399 INNER JOIN albums ON albums.albumid = tracks.albumid
400 GROUP BY
401 tracks.albumid
402 HAVING COUNT(trackid) > 15;
403
404 -- Get total length and bytes for each album
405 SELECT
406 -- Total length

```

Below the code is a results grid:

AlbumId	Title	COUNT(trackid)
1	Body Count	17
2	Penda Minha	18

And a status bar at the bottom:

Results: 41 rows returned in 20ms
At line 393:
SELECT
tracks.albumid,
title,

SQL_Seminar_3\A\ug0910.kip

```

402 HAVING COUNT(trackid) > 15;
403
404 -- Get total length and bytes for each album
405 SELECT
406   albumid,
407   sum(milliseconds) as length,
408   sum(bytes) as size
409 FROM
410   tracks
411 GROUP BY
412   albumid;
413
414 -- Get the album id, album title, maximum, minimum and the ave
415 SELECT
416   tracks.albumid,
417   title,
418   min(milliseconds)

```

Result: 41 rows returned in 20ms
At line 393:
SELECT
tracks.albumid,
title,

AlbumId	Title	COUNT(trackid)
1 18	Body Count	17
2 21	Prenda Minha	18

Type of data currently in cells: Text / Numeric
1 char(s)

Remote

Identity Commit Last modified Size

SQL_Seminar_3\A\ug0910.kip

```

411 GROUP BY
412   albumid;
413
414 -- Get the album id, album title, maximum, minimum and the ave
415 SELECT
416   tracks.albumid,
417   title,
418   min(milliseconds),
419   max(milliseconds),
420   round(avg(milliseconds), 2)
421 FROM
422   tracks
423 INNER JOIN albums ON albums.albumid = tracks.albumid
424 GROUP BY
425   tracks.albumid;
426
427 -- Get count of tracks by media type and genre

```

Result: 41 rows returned in 20ms
At line 393:
SELECT
tracks.albumid,
title,

AlbumId	Title	COUNT(trackid)
1 18	Body Count	17
2 21	Prenda Minha	18

Type of data currently in cells: Text / Numeric
1 char(s)

Remote

Identity Commit Last modified Size

SQL_Seminar_3\A\ug0910.kip

```

423 INNER JOIN albums ON albums.albumid = tracks.albumid
424 GROUP BY
425   tracks.albumid;
426
427 -- Get count of tracks by media type and genre
428 SELECT
429   mediatypeid,
430   genreid,
431   count(trackid)
432 FROM
433   tracks
434 GROUP BY
435   mediatypeid,
436   genreid;
437
438 -- find albums that have the total length greater than 60,000,!

```

Result: 41 rows returned in 20ms
At line 393:
SELECT
tracks.albumid,
title,

AlbumId	Title	COUNT(trackid)
1 18	Body Count	17
2 21	Prenda Minha	18

Type of data currently in cells: Text / Numeric
1 char(s)

Remote

Identity Commit Last modified Size

Top 5 customers by sales?

The screenshot shows a SQL Server Management Studio window with a query editor and a results grid.

```

1 SELECT
2     c.CustomerId,
3     c.FirstName,
4     sum(items.UnitPrice * items.Quantity) as Sales
5 FROM
6     customers c
7     INNER JOIN
8         invoices i ON c.CustomerId = i.CustomerId
9     INNER JOIN
10        invoice_items items ON i.InvoiceId = items.InvoiceId
11    group by
12        c.CustomerId,
13        c.FirstName
14    order by 3 desc
15

```

Results grid:

CustomerID	FirstName	Sales
1	Helena	49.62
2	Richard	47.62

Message bar at the bottom:

Result: 19 rows returned in 1ms
At line 1:
SELECT
c.CustomerId,
c.FirstName,

SQL Subqueries

The screenshot shows a SQL Server Management Studio window with a query editor and a results grid.

```

450 -- SQL Subquery
451 -- Find all the tracks in the album with the title Let There Be Rock
452
453
454 SELECT trackid,
455     name,
456     tracks.albumid
457     FROM tracks left join albums
458     on tracks.albumid = albums.albumid
459     where albums.Title = 'Let There Be Rock';
460
461 --OR
462 SELECT trackid,
463     name,
464     albumid
465     FROM tracks
466     WHERE albumid = /

```

Results grid:

AlbumId	Title	COUNT(trackid)
1	Body Count	17
2	Prende Minha	18

Message bar at the bottom:

Result: 41 rows returned in 2ms
At line 395:
SELECT
trackid,
name,
tracks.albumid,
title,

The screenshot shows a SQL Server Management Studio window with a query editor and a results grid.

```

459 WHERE albums.Title = 'Let There Be Rock';
460
461 --OR
462 SELECT trackid,
463     name,
464     albumid
465     FROM tracks
466     WHERE albumid in (
467         SELECT albumid
468             FROM albums
469             WHERE title = 'Let There Be Rock'
470     );
471
472 -- Get the tracks that belong to the artist id = 12
473
474 SELECT
475     +
476

```

Results grid:

TrackId	Name	AlbumId
1	Go Deew	4
2	Dog Eat Dog	4

Message bar at the bottom:

Result: 0 rows returned in 0ms
At line 423:
SELECT trackid,
name,
albumid

SQL_Sesson_2Aug2011.mq TopCustomers.mq

```

468     FROM albums
469      WHERE title = 'Let There Be Rock'
470    );
471
472 -- Get the tracks that belong to the artist id = 12
473
474 SELECT
475   trackid,
476   name,
477   albumid
478   FROM
479   tracks
480   WHERE
481     albumid IN (
482       SELECT
483         albumid
484         FROM
485           tracks
486           WHERE
487             artistid = 12
488   );
489
490 -- Find the customers whose sales representatives are in Canada
491 SELECT customerid,
492   firstname,
493   lastname
494   FROM customers
495   WHERE supportrepid IN (
496     SELECT employeeid
497       FROM employees
498       WHERE country = 'Canada'
499   );
500
501 --OR
502 SELECT a.customerid,
503   a.firstname,
504   a.lastname
505   FROM customers AS a LEFT JOIN employees AS b
506     ON a.supportrepid = b.employeeid
507   WHERE b.country = 'Canada';

```

Results: 8 rows returned in 0ms
At line 442:
SELECT trackid,
 name,
 albumid

TrackId	Name	AlbumId
1	15 Go Down	4
2	16 Dog Eat Dog	4

Type of data currently in cells: Text / Numeric
8 char(8)

Remote

Name	Commit	Last modified	Size
------	--------	---------------	------

SQL_Sesson_2Aug2011.mq TopCustomers.mq

```

477   albumid
478   FROM
479   tracks
480   WHERE
481     albumid IN (
482       SELECT
483         albumid
484         FROM
485           albums
486           WHERE
487             artistid = 12
488   );
489
490 -- Find the customers whose sales representatives are in Canada
491 SELECT customerid,
492   firstname,
493   lastname
494   FROM customers
495   WHERE supportrepid IN (
496     SELECT employeeid
497       FROM employees
498       WHERE country = 'Canada'
499   );
500
501 --OR
502 SELECT a.customerid,
503   a.firstname,
504   a.lastname
505   FROM customers AS a LEFT JOIN employees AS b
506     ON a.supportrepid = b.employeeid
507   WHERE b.country = 'Canada';

```

Results: 8 rows returned in 0ms
At line 442:
SELECT trackid,
 name,
 albumid

TrackId	Name	AlbumId
1	15 Go Down	4
2	16 Dog Eat Dog	4

Type of data currently in cells: Text / Numeric
8 char(8)

Remote

Name	Commit	Last modified	Size
------	--------	---------------	------

SQL_Sesson_2Aug2011.mq TopCustomers.mq

```

499 -- Find the customers whose sales representatives are in Canada
500 SELECT customerid,
501   firstname,
502   lastname
503   FROM customers
504   WHERE supportrepid IN (
505     SELECT employeeid
506       FROM employees
507       WHERE country = 'Canada'
508   );
509
510 --OR
511 SELECT a.customerid,
512   a.firstname,
513   a.lastname
514   FROM customers AS a LEFT JOIN employees AS b
515     ON a.supportrepid = b.employeeid
516   WHERE b.country = 'Canada';

```

Results: 8 rows returned in 0ms
At line 442:
SELECT trackid,
 name,
 albumid

TrackId	Name	AlbumId
1	15 Go Down	4
2	16 Dog Eat Dog	4

Type of data currently in cells: Text / Numeric
8 char(8)

Remote

Name	Commit	Last modified	Size
------	--------	---------------	------

SQLSession_25Aug2019.nq TopCustomers.udl

```

507     where b.country = 'Canada';
508
509
510 -- Get average album size in bytes
511 SELECT avg(album.size)
512   FROM (
513       SELECT sum(bytes) as size
514         FROM tracks
515        GROUP BY albumid
516    )
517   AS album;
518
519 -- Correlate Subquery?? Let's not cover in scope
520
521 -- INSERT ROWS in TABLES
522
523 -- single row

```

TrackId	Name	AlbumId
1 15	Go Down	4
2 16	Dog Eat Dog	4

Results: 2 rows returned in 0ms
At line 42:
SELECT trackid,
 name,
 albumid

Correlate subquery

SQLSession_25Aug2019.nq TopCustomers.udl

```

516     )
517   AS album;
518
519 -- Correlate Subquery?? Let's not cover in scope
520
521 -- INSERT ROWS in TABLES
522
523 -- single row
524 INSERT INTO artists (name)
525 VALUES
526 ('Bud Powell');
527
528 -- multiple rows
529 INSERT INTO artists (name)
530 VALUES
531 ('Buddy Rich'),
532 ('Candido'),
533 ('Charlie Byrd');
534
535 -- create backup of artists
536 CREATE TABLE artists_backup(
537   artistid INTEGER PRIMARY KEY AUTOINCREMENT,
538   name NVARCHAR
539 );
540 INSERT INTO artists_backup SELECT
541   artistid

```

TrackId	Name	AlbumId
1 15	Go Down	4
2 16	Dog Eat Dog	4

Results: 2 rows returned in 0ms
At line 42:
SELECT trackid,
 name,
 albumid

SQLSession_25Aug2019.nq TopCustomers.udl

```

525 VALUES
526 ('Bud Powell');
527
528 -- multiple rows
529 INSERT INTO artists (name)
530 VALUES
531 ('Buddy Rich'),
532 ('Candido'),
533 ('Charlie Byrd');
534
535 -- create backup of artists
536 CREATE TABLE artists_backup(
537   artistid INTEGER PRIMARY KEY AUTOINCREMENT,
538   name NVARCHAR
539 );
540 INSERT INTO artists_backup SELECT
541   artistid

```

TrackId	Name	AlbumId
1 15	Go Down	4
2 16	Dog Eat Dog	4

Results: 2 rows returned in 0ms
At line 42:
SELECT trackid,
 name,
 albumid

To insert values in a table

```

522 -- single row
523 INSERT INTO artists (name)
524 VALUES ('Inserted Artist');
525
526 -- multiple rows
527 INSERT INTO artists (name)
528 VALUES ('Buddy Rich'), ('Candido'), ('Charlie Byrd');
529
530 -- create backup of artists
531 CREATE TABLE artists backup(
532     artistid INTEGER PRIMARY KEY AUTOINCREMENT,
533     name NVARCHAR(40)
  
```

TrackId	Name	AlbumId
1 15	Go Down	4
2 16	Dog Eat Dog	4

Results: 0 rows returned in 1ms
At line 422
SELECT artistid,
 name,
 albumid

ArtistId	Name
257	Academy of St. M...
258	Les Arts Fluoréen...
259	The 12 Cellists of ...
260	Adrian Leaper & D...
261	Roger Norrington, ...
262	Charles Dutoit & L'...
263	Equule Brass Ensem...
264	Kent Nagano and ...
265	Julian Bream
266	Martin Roscoe
267	Göteborgs Symfon... Izhak Perlman
268	Michele Campanella
269	Gerald Moore
270	Mela Tenenbaum,...
271	Emerson String Q...
272	C. Monteverdi, Ing...
273	Nash Ensemble
274	Philip Glass Ensem...
275	Bud Powell
276	Buddy Rich
277	Candido
278	Charlie Byrd
279	Inserted Artist

Insert multiple values in a table

```

527 -- multiple rows
528 INSERT INTO artists (name)
529 VALUES ('Buddy Rich'), ('Candido'), ('Charlie Byrd');
530
  
```

Create new table – artists backup is table created from artist table

New rows can be inserted into new table from original tables

```

535 -- create backup of artists
536 CREATE TABLE artists backup(
537     artistid INTEGER PRIMARY KEY AUTOINCREMENT,
538     name NVARCHAR(40)
  );
539
540 INSERT INTO artists_backup SELECT
541     artistid,
542     name
543     FROM
544     artists;
  
```

Update values in a table

```
SQL_Sensor_54\sql2019.nf TopCustomers.nf

540 INSERT INTO artists_backup
541     SELECT
542         artistid,
543         name
544     FROM
545         artists;
546
547 -- SQL Update
548 -- Update the last name of Jane (emp id = 3) as she got married
549 UPDATE employees
550 SET lastname = 'Reddy'
551 WHERE
552     employeeid = 3;
553
554 /* Suppose Park Margaret locates in Toronto and you want to change
555 city, and state information.*/
556
557
558
559
560
```

Employee Data										
EmployeeID	Lastname	Firstname	Title	ReportsTo	Birthdate	HireDate	Address	City	State	Country
1	Adams	Andrew	General Manager	None	1962-02-18 00:00:00	2002-08-14 00:00:00	1112 Jasper Ave	Edmonton	AB	Canada
2	Edwards	Nancy	Sales Manager	1	1958-12-06 00:00:00	2002-05-01 00:00:00	825 8 Ave SW	Calgary	AB	Canada
3	Reddy	Jane	Sales Support Agent	2	1975-03-29 00:00:00	2002-04-15 00:00:00	1111 6 Ave SW	Calgary	AB	Canada
4	Park	Margaret	Sales Support Agent	2	1947-09-19 00:00:00	2003-05-03 00:00:00	683 10 Street SW	Toronto	ON	Canada
5	Johansen	Steve	Sales Support Agent	2	1965-03-03 00:00:00	2003-10-17 00:00:00	7729 41 Ave	Calgary	AB	Canada
6	Mitchell	Michael	IT Manager	1	1973-07-01 00:00:00	2003-10-17 00:00:00	5827 Bowrise Blvd.	Calgary	AB	Canada
7	King	Robert	IT Staff	6	1970-05-29 00:00:00	2004-01-02 00:00:00	590 Columbia Boulevard	Lethbridge	AB	Canada
8	Celentano	Laure	IT Staff	6	1968-01-09 00:00:00	2004-03-04 00:00:00	923 7 ST NW	Lethbridge	AB	Canada

```
SQL_Session_5Aug2019 TopCustomers
```

```
549 UPDATE employees
550 SET lastname = 'Reddy'
551 WHERE
552   employeeid = 3;
553
554 /* Suppose Park Margaret locates in Toronto and you want to change
555    city, and state information.*/
556
557 UPDATE employees
558 SET city = 'Toronto',
559     state = 'ON',
560     postalcode = 'M5P 2N7'
561 WHERE
562   employeeid = 4;
563
564 -- Deleting rows from table
565 DELETE
```

Results: query executed successfully. Took 0ms, 1 rows affected

```
1> USE employees
2> UPDATE employees
3> SET lastname = 'Reddy'
4> UPDATE employees
```

Deleting rows

The screenshot shows a SQL Server Management Studio interface. On the left, a query window displays T-SQL code for deleting rows from a backup table. The code includes conditions for employee ID, artist ID, and name. On the right, a results pane shows the output of the executed query, indicating 1 row affected.

```
561 WHERE
562     employeeid = 4;
563
564 -- Deleting rows from table
565 DELETE
566 FROM
567     artists_backup
568 WHERE
569     artistid = 1;
570     -- Delete based on condition
571 DELETE
572 FROM
573     artists_backup
574 WHERE
575     name LIKE '%Santana%';
576
577 -- Removing all rows of database
578 DELETE
579 FROM
580     artists_backup;
```

Result: query executed successfully. Took 1ms, 1 rows affected
At line 54:
DELETE
FROM
 artists_backup
WHERE
 name = 'Randy'

Delete all the rows and dropping a table

The screenshot shows a SQL Server Management Studio interface. A query window contains T-SQL code for deleting rows and dropping a table. The code specifies a WHERE clause for artist ID and a general delete statement. The results pane shows the successful execution of the query, with 270 rows affected.

```
568 WHERE
569     artistid = 280;
570     -- Delete based on condition
571 DELETE
572 FROM
573     artists_backup
574 WHERE
575     name LIKE '%Santana%';
576
577 -- Removing all rows of database
578 DELETE
579 FROM
580     artists_backup;
581
582 -- Dropping a table
583 drop table artists_backup;
```

Result: query executed successfully. Took 1ms, 270 rows affected
At line 578:
DELETE
FROM
 artists_backup;

Export table as csv

The screenshot shows a database interface with the following details:

- Left Panel (Tables):**
 - Tables (19)
 - OrderDetails
 - Products
 - PropertiesInfo
 - SubTable
 - albums
 - artists
 - books
 - categories
 - employees
 - genres
 - invoice_items
 - invoices
 - media_types
 - playlists
 - sample_table
 - sqlite_sequence
 - sqlite_stat1
 - tracks
 - tracks1 (1)
 - Indices (10)
 - IFC_AlbumArtistId
 - IFC_CustomerSupportRepId
 - IFC_EmployeeReportsTo
 - IFC_InvoiceCustomerId
 - IFC_InvoiceLineInvoiceId
 - IFC_PlaylistLineBackId
 - IFC_PlaylistTrackId
 - IFC_SeqAlbumId
 - IFC_TrackGeneral
 - IFC_TrackMediaTypeId
 - Views (0)
 - Triggers (0)
- Context Menu (employees):**
 - Browse Table
 - Modify Table
 - Delete Table
 - Copy Create statement
 - Export as CSV file
- Edit Database Cell Window:**

Mode: Text

2015-01-01

Type of data currently in cell: Text / Numeric
as char(10)
- Remote Window:**

Identity

Name	Commit	Last modified	Size