

<https://www.w3schools.com/SQL/>

SQL Primary Key

The PRIMARY KEY uniquely identifies each record in a table.

For e.g : Store ID in Stores Table, Order ID in Sales Table

SQL Foreign Key

A FOREIGN KEY is a field/column (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

For e.g Suppose we have two tables Stores Table and Customers Table. Stores Table contains details of all Reliance stores across India and Customers table having details of the all customers. Store ID is present in both the tables but Store ID will be unique in Stores Table and it can be present against multiple customers in customers table as many people are visiting one store.

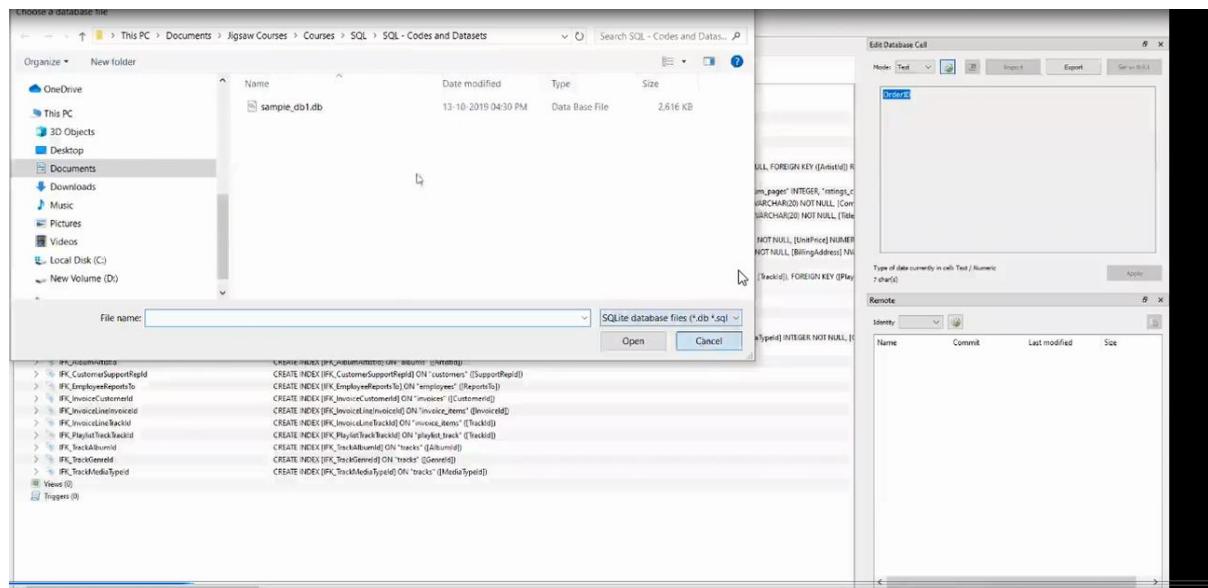
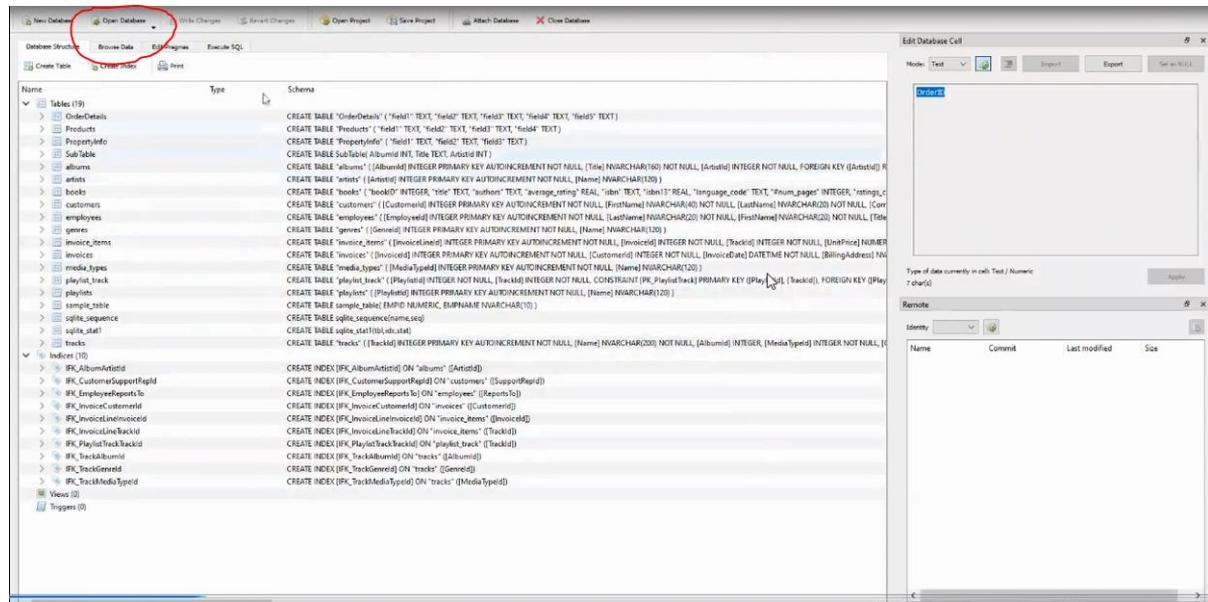
So, Store ID will be a Primary key in Stores Table and Foreign Key in Customers Table and two tables can be joined with Store ID.

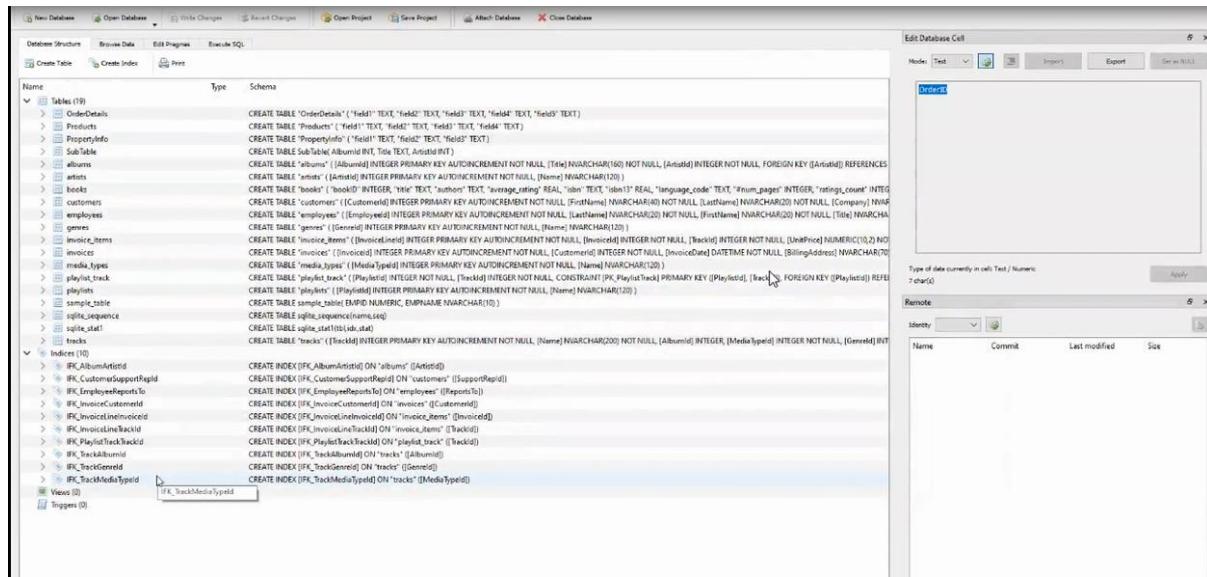
Primary key vs Unique Key

In the case of a primary key, we cannot save NULL values. In the case of a unique key, we can save a null value, however, only one NULL value is supported.

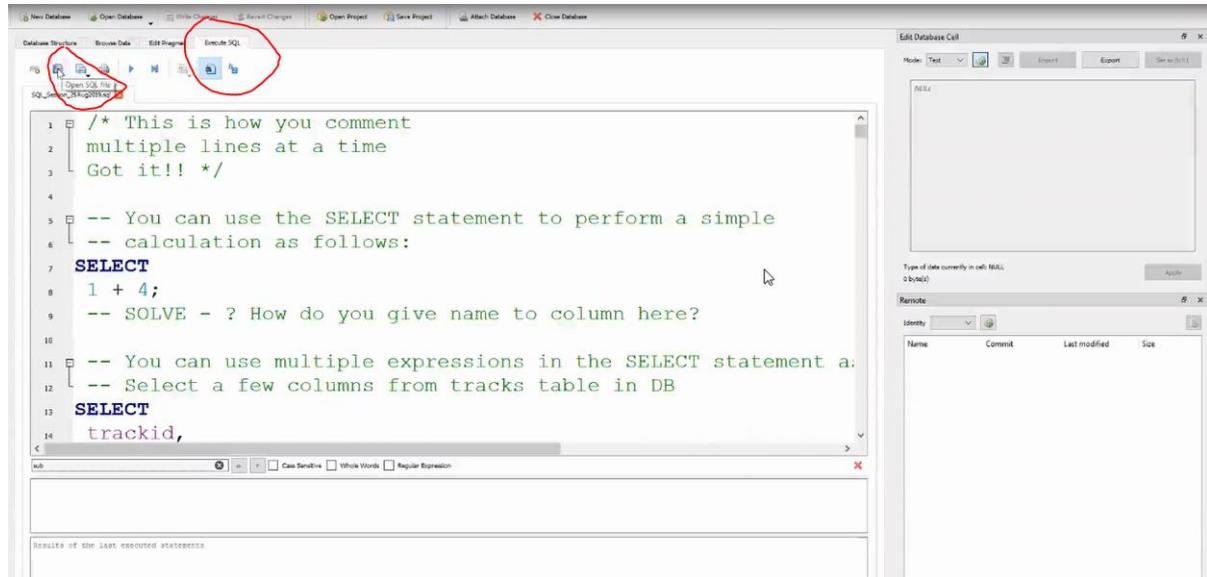
<https://byjus.com/gate/difference-between-primary-key-and-unique-key/>

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





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 the SQL Server Management Studio interface. The top window displays a T-SQL script with several comments and a simple SELECT statement. The bottom window shows the execution results for the query `1 + 4`, which returned a single row with the value 5. On the right, there is a file browser window showing a folder named 'Remote'.

```
SQL_Sessions_SQLServer.kip
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
<-->
1 + 4
5
Results: 1 rows returned in 1ms
At line 7:
SELECT
1 + 4;
```

Columnname is available with this piece of code

The screenshot shows the Microsoft SQL Server Management Studio (SSMS) interface. On the left, a query window displays a T-SQL script. The script includes comments on commenting multiple lines, using SELECT for calculations, and using multiple expressions. It also shows how to select specific columns from the tracks table. A cursor is positioned over the word 'column' in the third line of the script. Below the script, there are search and filter options for the results grid. On the right, a results grid is visible with one row containing the value 'NULL'. The status bar at the bottom shows the result count as 1 row returned.

```
/* This is how you comment
multiple lines at a time
Got it!! */

-- You can use the SELECT statement to perform a simple
-- calculation as follows:
SELECT
    1 + 4 as columnName;
-- 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
```

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

Trackid table

The screenshot shows the MySQL Workbench interface with the 'Album' table selected in the left sidebar under the 'Tables (1)' section. The table structure is displayed in the main pane:

```
CREATE TABLE `Album` (
    `ArtistId` INT NOT NULL,
    `Title` NVARCHAR(160) NOT NULL,
    `AlbumId` INT NOT NULL,
    `MediaTypeId` INT NOT NULL,
    `GenreId` INT NOT NULL,
    `LabelId` INT NOT NULL,
    `TrackCount` INT NOT NULL,
    `UnitPrice` NUMERIC(10, 2) NOT NULL,
    PRIMARY KEY (`AlbumId`),
    FOREIGN KEY (`ArtistId`) REFERENCES `Artist`(`ArtistId`),
    FOREIGN KEY (`MediaTypeId`) REFERENCES `MediaType`(`MediaTypeId`),
    FOREIGN KEY (`GenreId`) REFERENCES `Genre`(`GenreId`),
    FOREIGN KEY (`LabelId`) REFERENCES `Label`(`LabelId`)
)
```

Relationships are shown with red lines connecting the 'ArtistId' column in the 'Album' table to the 'Artist' table, and the 'MediaTypeId' and 'GenreId' columns to their respective tables. A red box highlights the 'Artist' table entry.

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

	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
1	1	For Those About ...	1	1	1	Angus Young, Mal...	343719	11170334	0.99
2	2	Balls To The Wall	2	1	1		342562	5510424	0.99
3	3	Fast As A Shark	3	2	1			3909094	0.99
4	4	Restless And Wild	3	2	1	P. Bonham, K.C. Sh...	232091	4331779	0.99
5	5	Princess Of The De...	3	2	1	Dossey & R.A. Smi...	37518	6290521	0.99
6	6	Put The Finger On...	1	1	1	Angus Young, Mal...	205662	6713451	0.99
7	7	Let's Get It Up	1	1	1	Angus Young, Mal...	233926	7636561	0.99
8	8	Inject The Venom	1	1	1	Angus Young, Mal...	210834	6852860	0.99
9	9	Snowballed	1	1	1	Angus Young, Mal...	203102	6599424	0.99
10	10	Evil Walks	1	1	1	Angus Young, Mal...	263497	8611245	0.99
11	11	C.O.D.	1	1	1	Angus Young, Mal...	199836	6566314	0.99
12	12	Breeding The Rules	1	1	1	Angus Young, Mal...	263288	8596840	0.99
13	13	Night Of The Long...	1	1	1	Angus Young, Mal...	205688	6706347	0.99
14	14	Spellbound	1	1	1	Angus Young, Mal...	270863	8817038	0.99
15	15	Go Down	4	1	1	AC/DC	331180	10847611	0.99
16	16	Dog Eat Dog	4	1	1	AC/DC	215196	7032162	0.99
17	17	Let There Be Rock	4	1	1	AC/DC	366654	12021261	0.99
18	18	Bad Boy Boogie	4	1	1	AC/DC	267728	8776140	0.99
19	19	Problem Child	4	1	1	AC/DC	325041	10617116	0.99
20	20	Overdose	4	1	1	AC/DC	369319	12066294	0.99
21	21	Hell Ain't A Bad Pl...	4	1	1	AC/DC	254380	8331286	0.99
22	22	Whole Lotta Rosie	4	1	1	AC/DC	323761	10547154	0.99
23	23	Walk On Water	5	1	1	Steven Tyler, Joe ...	295680	9719579	0.99
24	24	Love In An Elevator	5	1	1	Steven Tyler, Joe ...	321828	10552051	0.99

Data can be filtered over here

	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
1	1	For Those About ...	1	1	1	Angus Young, Mal...	343719	11170334	0.99
2	6	Put The Finger On...	1	1	1	Angus Young, Mal...	205662	6713451	0.99
3	7	Let's Get It Up	1	1	1	Angus Young, Mal...	233926	7636561	0.99
4	8	Inject The Venom	1	1	1	Angus Young, Mal...	210834	6852860	0.99
5	9	Snowballed	1	1	1	Angus Young, Mal...	203102	6599424	0.99
6	10	Evil Walks	1	1	1	Angus Young, Mal...	263497	8611245	0.99
7	11	C.O.D.	1	1	1	Angus Young, Mal...	199836	6566314	0.99
8	12	Breeding The Rules	1	1	1	Angus Young, Mal...	263288	8596840	0.99
9	13	Night Of The Long...	1	1	1	Angus Young, Mal...	205688	6706347	0.99
10	14	Spellbound	1	1	1	Angus Young, Mal...	270863	8817038	0.99

The screenshot shows the tracks table in the Database Structure pane. The table has columns: TrackId, Name, AlbumId, MediaTypeId, GenreId, Composer, Milliseconds, Bytes, and UnitPrice. The first row is selected. An 'Edit Database Cell' dialog is open on the right, showing the value '1' in the cell for the first column. The dialog also includes fields for Mode (Text), Import, Export, and Set to NULL.

This screenshot is similar to the one above, but the 'Edit Database Cell' dialog now contains the value '101'. The rest of the interface is identical, showing the tracks table with its data and the standard SSMS toolbar.

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

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

```

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? */

```

The execution results pane below shows two rows of data from the tracks table:

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 Wall	NULL	0.99

At the bottom, the status bar indicates: Result: 3803 rows returned in 11ms. At line 12: -- Select a few columns from tracks table in DB.

Addition of two columns

The screenshot shows a SQL Server Management Studio interface. In the top-left window, a script named 'SQL_Sesson_15Aug2019.sql' is open 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
14     SELECT
15         trackid,
16         name,
17         composer,
18         unitprice,
19         Milliseconds + Bytes as name
20
21     FROM
22         tracks;
23
24     /* SOLVE- ? How to select All the columns from tracks table?
25     - Way/s - Which one is easier and which is recommended? */

```

In the bottom-right window, a results grid displays two rows of data from the 'tracks' table:

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		0.99

Below the results grid, the status bar shows: Result: 3503 rows returned in 11ms At line 12: -- Select a few columns from tracks table in DB

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 interface. In the top-left window, a script named 'SQL_Sesson_15Aug2019.sql' is open with the following code:

```

16     composer,
17     unitprice
18
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     -- Get the cities from where customer belongs
26
27     SELECT
28         city
29     FROM
30         customers;
31
32     -- SOLVE - ? What is the issue with above query?

```

In the bottom-right window, a results grid displays two rows of data from the 'customers' table:

City
São José dos Campos
Stuttgart

Below the results grid, the status bar shows: Result: 59 rows returned in 1ms At line 26: SELECT
 CITY
FROM

Distinct will select only unique cities (53 rows)

The screenshot shows a SQL Server Management Studio interface. In the top-left window, a query is being typed:

```
28 FROM
29 customers;
-- SOLVE - ? What is the issue with above query?
31
32
33 -- Use of DISTINCT
34 SELECT DISTINCT
35 city
36 FROM
37 customers;
```

Below the query, a note says: "SOLVE - ? Get the companies of the customers -- What is unusual about result?"

At the bottom of the query window, there is a results grid titled "City" with two rows:

City
1 São José dos Campos
2 Stuttgart

Below the results grid, the status bar shows: "Result: 53 rows returned in 1ms At line 34: SELECT DISTINCT city FROM".

Orderby will sort the data

The screenshot shows a SQL Server Management Studio interface. In the top-left window, a query is being typed:

```
58 FROM
59 tracks;
60
61 -- Now get the same data sorted based on AlbumId column in asc
62 SELECT
63 name,
64 milliseconds,
65 albumid
66 FROM
67 tracks
68 ORDER BY
69 albumid ASC;
```

Below the query, a note says: "Now sort the sorted result (by AlbumId) above... by theMilliseconds column in descending order"

At the bottom of the query window, there is a results grid titled "City" with two rows:

City
1 São José dos Campos
2 Stuttgart

Below the results grid, the status bar shows: "Result: 53 rows returned in 1ms At line 34: SELECT DISTINCT city FROM".

ASC : Ascending ; DESC : Descending

The screenshot shows a SQL Server Management Studio interface. In the top-left window, a query is being typed:

```
67 tracks
68 ORDER BY
69 albumid ASC;
```

Below the query, a note says: "Now sort the sorted result (by AlbumId) above... by theMilliseconds column in descending order"

At the bottom of the query window, there is a results grid titled "City" with two rows:

City
1 São José dos Campos
2 Stuttgart

Below the results grid, the status bar shows: "Result: 53 rows returned in 1ms At line 34: SELECT DISTINCT city FROM".

Using column positions to sort

The screenshot shows the MySQL Workbench interface. The main area contains a SQL editor window with the following 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
```

Below the editor is a toolbar with various icons. The results pane at the bottom displays a table with two rows:

City
São José dos Campos
Stuttgart

The status bar at the bottom indicates "Result: 83 rows returned in 0ms".

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

```
-- Use of LIMIT clause
-- get the first 10 rows in the tracks table
SELECT
    trackId,
    name
FROM
    tracks
LIMIT 10;

-- get 10 rows starting from the 10th row in the tracks table
SELECT
    trackId,
    name
FROM
    tracks
LIMIT 10 OFFSET 10;
```

sort and limit on single piece of code

SQLSession_5Aug2015

```
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 whe.  
127 -- select customerid, firstname, company  
   from customers
```

Sub

TrackId Name

1	11	C.O.D.
2	12	Breaking The Rules

Results: 10 rows returned in 3ms
At time 103:
-- 2015-08-05 10:45:20 row starting from the 10th row in the tracks table
SELECT

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

The screenshot shows a SQL session window titled "SQLSession_25Aug2019.mdf". The query pane contains the following code:

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

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

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

SELECT
```

The results pane shows a table with two rows:

TrackId	Name
1	11 C.O.D.
2	12 Breaking The Rules

The status bar at the bottom indicates: Result: 10 rows returned in 3ms At line 127: select customerid, firstname, company from customers where company is NULL;

Where statement is used in this piece of code

The screenshot shows a SQL session window titled "SQLSession_25Aug2019.mdf". The query pane contains the following code:

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

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

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

SELECT
```

The results pane shows a table with two rows:

Customerid	Firstname	Company
48	Manoj	NULL
49	Puja	NULL

The status bar at the bottom indicates: Result: 49 rows returned in 1ms At line 127: select customerid, firstname, company from customers where company is NULL;

The screenshot shows a SQL session window titled "SQLSession_25Aug2019.mdf". The query pane contains the following code:

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

133 SELECT
134     name,
135     milliseconds,
136     bytes,
137     albumid
138 FROM
139     tracks
140 WHERE
141     albumid = 1;

-- SOLVE: Get tracks on the album 1 that have the length great
```

The results pane shows a table with two rows:

Customerid	Firstname	Company
35	Ladislav	NULL
36	Hugh	NULL

The status bar at the bottom indicates: Result: 49 rows returned in 1ms At line 127: select customerid, firstname, company from customers where company is NULL;

And will apply to conditions inside where statement

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

```
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 -- Find which tracks are composed by all people with 'Smith' in  
-- wildcard characters
```

The results grid shows two rows:

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

Details: 4 rows returned in 0ms
At line 148:
SELECT
 name,
 milliseconds,

The screenshot shows a SQL Server Management Studio window with a query editor and a results grid. The query is identical to the one above:

```
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 -- Find which tracks are composed by all people with 'Smith' in  
-- wildcard characters
```

The results grid shows two rows:

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

Details: 4 rows returned in 0ms
At line 148:
SELECT
 name,
 milliseconds,

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

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

```
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 <= , =
```

A red bracket highlights the comparison operators in the WHERE clause: >, >=, <, <=, and =.

The results grid shows two rows:

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

Details: 4 rows returned in 0ms
At line 148:
SELECT
 name,
 milliseconds,

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_Sesson_25Aug2019.edb

```
160
161 -- Find which tracks are composed by all people with 'Smith' in
162 -- Wildcard characters
163 SELECT
164     name,
165     albumid,
166     composer
167 FROM
168     tracks
169 WHERE
170     composer LIKE '%Smith%'
171 ORDER BY
172     albumid;
```

-- Use of wildcards % and _

```
175 SELECT
176     name
177 
```

Results: 97 rows returned in 0ms
At line 163:
SELECT
 name,
 albumid,

Name	AlbumId	Composer
Killing Floor	19	Adrian Smith
Machete Man	19	Adrian Smith

SQL_Sesson_25Aug2019.edb

```
172
173 albumid;
174
175 -- Use of wildcards % and _
176 SELECT
177     trackid,
178     name
179 FROM
180     tracks
181 WHERE
182     name LIKE 'Wild%';
183
184 SELECT
185     trackid,
186     name
187 FROM
188     tracks
189 WHERE
190     name LIKE 'Wild%';
```

Results: 5 rows returned in 0ms
At line 175:
SELECT
 trackid,
 name

TrackId	Name
1245	Widget Dreams
1973	Wild Side

SQL_Sesson_25Aug2019.edb

```
175
176 SELECT
177     trackid,
178     name
179 FROM
180     tracks
181 WHERE
182     name LIKE 'Wild%';
183
184 SELECT
185     trackid,
186     name
187 FROM
188     tracks
189 WHERE
190     name LIKE '%Wild';
```

Results: 5 rows returned in 0ms
At line 175:
SELECT
 trackid,
 name

TrackId	Name
1245	Wildest Dreams
1973	Wild Side

Or condition inside where

```
SQL_Sesson_5Aug2019.mpx | SQL

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
< >    Case Sensitive  Whole Words  Regular Expression
x
sub

|   | Name              | AlbumId | MediaTypeId |
|---|-------------------|---------|-------------|
| 1 | Balls to the Wall | 2       | <b>2</b>    |
| 2 | Fast As a Shark   | 3       | <b>2</b>    |


x
Result: 451 rows returned in 37ms
At line 203:
SELECT
    name,
    albumid,
    mediatypeid
```

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

```
SQL_Sessions_5Aug2019.sql

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
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
570
```

Not in condition – opposite of in

```
SQL_Session_25Aug2019.edb
```

```
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 WHERE
232 genreid NOT IN (1,2,3);
233
234
235 -- Use of BETWEEN operator
236 -- finds invoices whose total is between 14.01 and 18.96.
<-->
```

SQL Server Management Studio interface showing the execution results:

- Output pane (top right): Shows the result of the query: "Type of data currently in cell: NULL" and "0 byte(s)".
- Properties pane (bottom right): Shows the connection details: "Remote" and "Identity".
- Results pane (bottom left): Shows the results of the query, which are empty.
- Object Explorer pane (bottom center): Shows the database structure with tables like "Albums", "Genre", "MediaTypes", and "Tracks".

Between condition

```
SQL_Session_15Aug2019.sql

222 genereid NOT IN (1,2,3);
223
224
225 -- Use of BETWEEN operator
226 -- finds invoices whose total is between 14.91 and 18.86:
227
228 SELECT
229     invoiceId,
230     BillingAddress,
231     Total
232
233 FROM
234     invoices
235 WHERE
236     Total BETWEEN 14.91 and 18.86
237 ORDER BY
238     Total;
239
240
241
242
243
244
245
246
247
```

Result: 451 rows returned in 42ms
At line 217:
SELECT
 genereid,
 name, albums, mediatypes
FROM

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

notbetween condition

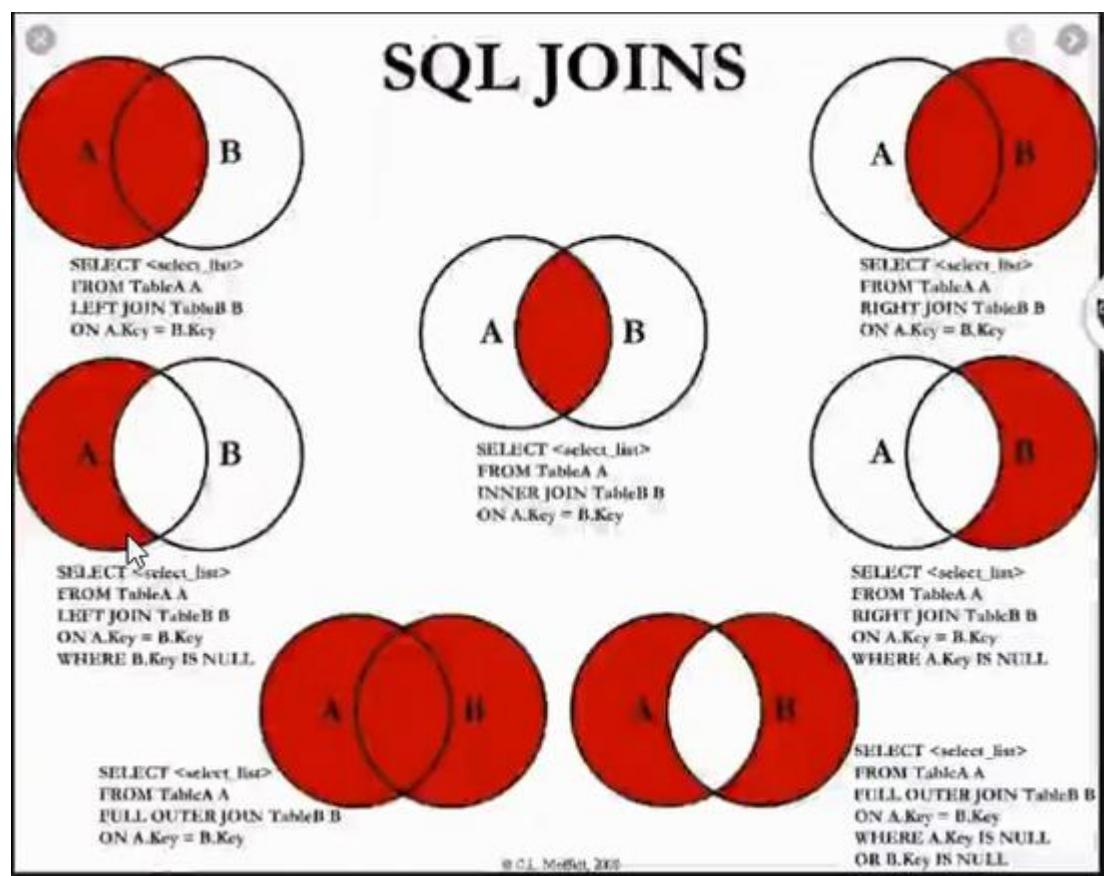
```
SQL_Sesson_35\ug2019.sql
```

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

Between condition on dates

```
SQL_Lesson_3A.qsparks
265
266
267 -- finds invoices whose invoice dates are from January 1 2010 .
268 SELECT
269     InvoiceId,
270     BillingAddress,
271     InvoiceDate,
272     Total
273 FROM
274     invoices
275 WHERE
276     InvoiceDate BETWEEN '2010-01-01' AND '2010-01-31'
277 ORDER BY
278     InvoiceDate;
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
570
```

SQL Joins



Cust_id City		Cust_id Sales		Inner Join		
1	2	1	2	Cust_id	Sales	City
1	Mumbai	1	100	1	100	Mumbai
2	Mumbai	2	220.5	2	220.5	Mumbai
3	Bangalore	4	50	4	50	Mysore
4	Mysore	5	400	5	400	Hyderabad
5	Hyderabad	7	30			
6	Pune	8	480.7			

Left Join			Right Join			Full Outer Join		
Cust_id	City	Sales	Cust_id	Sales	City	Cust_id	Sales	City
1	Mumbai	100	1	100	Mumbai	1	100	Mumbai
2	Mumbai	220.5	2	220.5	Mumbai	2	220.5	Mumbai
3	Bangalore	NULL	4	50	Mysore	3	50	Bangalore
4	Mysore	50	5	400	Hyderabad	4	400	Mysore
5	Hyderabad	400	7	30	NULL	5	400	Hyderabad
6	Pune	NULL	8	480.7	NULL	6	0	Pune
						7	30	NULL
						8	480.7	NULL

Inner join on tracks and album

Trackid and name from tracks table

Title from album table

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

```
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 editor is a results grid titled "How do you validate join is correct?". The grid has columns: TrackId, Name, and Title. It contains two rows of data:

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

At the bottom of the results grid, it says "Result: 3503 rows returned in 87ms".

The top-right pane shows a "REMOTE" connection status window.

Usage of alias for table names

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

```
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 editor is a results grid titled "How do you validate join is correct?". The grid has columns: TrackId, Name, and Title. It contains two rows of data:

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

At the bottom of the results grid, it says "Result: 3503 rows returned in 87ms".

The top-right pane shows a "REMOTE" connection status window.

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

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

The screenshot shows the SQL query window with the code above. Below it is the results grid:

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

Result: 3503 rows returned in 13ms
At line 304:
SELECT
 trackid,
 name,

On the right, there is a 'Results' pane showing a table with columns Name, Commit, Last modified, and Size.

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 SQL query window with the code above. Below it is the results grid:

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

Result: 3503 rows returned in 13ms
At line 304:
SELECT
 trackid,
 name,

On the right, there is a 'Results' pane showing a table with columns Name, Commit, Last modified, and Size.

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

The screenshot shows the SQL query window with the code above. Below it is the results grid:

TrackId	Track	Album	Artist
1	123	Quadrant	The Best Of Billy Cobham
2	124	Snoopy's search-Red Baron	The Best Of Billy Cobham
3	125	Spanish moss-"A sound portrait"-Spanish moss	The Best Of Billy Cobham
4	126	Moon germs	The Best Of Billy Cobham
5	127	Stratus	The Best Of Billy Cobham
6	128	The pleasant pheasant	The Best Of Billy Cobham
7	129	Solo-Pianohandler	The Best Of Billy Cobham
8	130	Do what cha wanna	The Best Of Billy Cobham

Result: 8 rows returned in 10ms
At line 322:
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;

On the right, there is a 'Results' pane showing a table with columns Name, Commit, Last modified, and Size.

```

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

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

SELECT
artists.ArtistId,
albumId
FROM
artists
LEFT JOIN albums ON albums.artistid = artists.artistid
ORDER BY
albumid;

--OR
SELECT
  artist_id, artist_name

```

Result: 422 rows returned in 2ms
At line 335:
SELECT
 artist_id,
 albumId

ArtistId	AlbumId
53	NULL
54	NULL

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 -- Note - SQLite does not support RIGHT JOIN and FULL OUTER jo
358
359

```

ArtistId	AlbumId
53	NULL
54	NULL

Groupby (like pivot table in excel)

```

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

```

Result: 241 rows returned in 3ms
At line 361:
SELECT
 albumid,
 COUNT(trackid) AS track_count

AlbumId	track_count
1	10
2	1

Sorting on top of groupby

SQL Session_25Aug2019.mdf

```

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
  
```

Results grid:

AlbumId	track_count
1	57
2	34

SQL Server Management Studio interface showing the results grid and the query editor.

SQL Session_25Aug2019.mdf

```

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 select a.albumID , b.title, count (trackid) as track_count
383 from
384 tracks as a
385 left join
386 albums as b on a.albumid = b.albumid
387 group by 1,2;
388
389 -- GROUP BY with HAVING clause - to filter by aggregate value
390 -- get the albums that have more than 15 tracks
  
```

Results grid:

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

SQL Server Management Studio interface showing the results grid and the query editor.

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

SQL Session_25Aug2019.mdf

```

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 albumid
  
```

Results grid:

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

SQL Server Management Studio interface showing the results grid and the query editor.

SQLSession_15Aug2019.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 average
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
428
```

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

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

SQLSession_15Aug2019.kip

```
411 GROUP BY
412     albumid;
413
414 -- Get the album id, album title, maximum, minimum and the average
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
428
```

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

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

SQLSession_15Aug2019.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,!
439
```

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

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

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	6	Helena	49.62
2	26	Richard	47.62

Message bar: Result: 59 rows returned in 2ms
At line 1:
26
c.CustomerId,
c.FirstName,

SQL Subqueries

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

```

450
451 -- SQL Subquery
452 -- Find all the tracks in the album with the title Let There Be Rock
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	Prenda Minha	18

Message bar: Result: 41 rows returned in 2ms
At line 393:
SELECT
trackid,
name,
albumid
FROM tracks
WHERE albumid = /

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
+ trackid

```

Results grid:

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

Message bar: Result: 8 rows returned in 3ms
At line 442:
SELECT trackid,
name,
albumid

```
FROM albums
WHERE title = 'Let There Be Rock'
);

-- Get the tracks that belong to the artist id = 12

SELECT
trackid,
name,
albumid
FROM
tracks
WHERE
albumid IN (
    SELECT
    albumid
    FROM
    albums
    WHERE
    artistid = 12
);

Result: 0 rows returned in 0ms
At line 424:
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
\$ chart()

Apply

Remote

Identity

Name Commit Last modified Size

```
albumid
FROM
tracks
WHERE
albumid IN (
    SELECT
    albumid
    FROM
    albums
    WHERE
    artistid = 12
);

-- Find the customers whose sales representatives are in Canada
SELECT customerid,
firstname,
lastname
```

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

Result: 0 rows returned in 0ms
At line 424:
SELECT
trackid,
name,
albumid

Type of data currently in cells: Text / Numeric
\$ chart()

Apply

Remote

Identity

Name Commit Last modified Size

```
-- Find the customers whose sales representatives are in Canada
SELECT c.customerid,
firstname,
lastname
FROM customers
WHERE supportrepid IN (
    SELECT employeeid
    FROM employees
    WHERE country = 'Canada'
);

--OR
SELECT a.customerid,
a.firstname,
a.lastname
FROM customers AS a LEFT JOIN employees AS b
```

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

Result: 0 rows returned in 0ms
At line 424:
SELECT
trackid,
name,
albumid

Type of data currently in cells: Text / Numeric
\$ chart()

Apply

Remote

Identity

Name Commit Last modified Size

```
SQLSession_25Aug2019.nq TopCustomers.nq
```

```
507      where b.country = 'Canada';  
508  
509  
510      -- Get average album size in bytes  
511      SELECT avg(album.size)  
512      FROM (          SELECT sum(bytes) as size  
513          FROM tracks  
514          GROUP BY albumid  
515      )  
516      AS album;  
517  
518      -- Correlate Subquery?? Let's not cover in scope  
519  
520      -- INSERT ROWS in TABLES  
521  
522      -- single row  
523      INSERT INTO artists (name)  
524      VALUES  
525      ('Bud Powell');  
526  
527      -- multiple rows  
528      INSERT INTO artists (name)  
529      VALUES  
530      ('Buddy Rich'),  
531      ('Candido'),  
532      ('Charlie Byrd');  
533  
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  
542      FROM artists;
```

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

Result: 2 rows returned in 0ms
At line 442:
SELECT trackid,
 name,
 albumid

```
Info
```

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

Remote

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

Correlate subquery

```
SQLSession_25Aug2019.nq TopCustomers.nq
```

```
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');
```

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

Result: 2 rows returned in 0ms
At line 442:
SELECT trackid,
 name,
 albumid

```
Info
```

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

Remote

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

```
SQLSession_25Aug2019.nq TopCustomers.nq
```

```
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  
542      FROM artists;
```

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

Result: 2 rows returned in 0ms
At line 442:
SELECT trackid,
 name,
 albumid

```
Info
```

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

Remote

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

To insert values in a table

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

```

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)
  
```

The results grid shows the inserted rows:

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

Below the results grid, the status bar indicates:

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

Oracle Database Navigator interface showing a table named 'artists' with the following data:

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

Insert multiple values in a table

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

```

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

The results grid shows the inserted rows:

TrackId	Name	AlbumId
1	Buddy Rich	4
2	Candido	4
3	Charlie Byrd	4

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

New rows can be inserted into new table from original tables

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

```

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;
  
```

The results grid shows the inserted rows:

ArtistId	Name
15	Go Down
16	Dog Eat Dog

Update values in a table

The screenshot shows the SQL Server Management Studio interface. In the center pane, there is a code editor window containing the following SQL script:

```

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 SELECT
558     artistid,
559     name
560
561

```

Below the code editor, the status bar indicates: "Result: query executed successfully. Took 1ms, 280 rows affected".

The screenshot shows the SQL Server Object Explorer interface. On the left, there is a tree view of database objects. In the center, there is a grid-based table viewer for the "employees" table. The table has the following columns: EmployeeId, LastName, FirstName, Title, ReportsTo, BirthDate, HireDate, Address, City, State, and Country. The data for the first few rows is as follows:

EmployeeId	LastName	FirstName	Title	ReportsTo	BirthDate	HireDate	Address	City	State	Country
1	Adams	Andrew	General Manager	NULL	1962-02-18 00:00:00	2002-08-14 00:00:00	1112 Jasper Ave N	Edmonton	AB	Canada
2	Edwards	Nancy	Sales Manager	1	1958-12-08 00:00:00	2002-05-01 00:00:00	835 B Ave SW	Calgary	AB	Canada
3	Reddy	Jani	Sales Support Agent	2	1973-09-29 00:00:00	2002-04-01 00:00:00	1111 A Ave SW	Calgary	AB	Canada
4	Park	Margaret	Sales Support Agent	2	1947-09-19 00:00:00	2003-03-03 00:00:00	663 10 Street SW	Toronto	ON	Canada
5	Johnson	Steve	Sales Support Agent	2	1965-03-03 00:00:00	2003-10-17 00:00:00	7727B 41 Ave	Calgary	AB	Canada
6	Mitchell	Michael	IT Manager	1	1973-07-01 00:00:00	2003-10-17 00:00:00	5827 Bowness Rd	Calgary	AB	Canada
7	King	Robert	IT Staff	6	1970-09-29 00:00:00	2004-01-02 00:00:00	590 Columbia Blvde	Lethbridge	AB	Canada
8	Callahan	Laura	IT Staff	6	1968-01-09 00:00:00	2004-03-04 00:00:00	923 7 ST NW	Lethbridge	AB	Canada

The screenshot shows the SQL Server Management Studio interface. In the center pane, there is a code editor window containing the following SQL script:

```

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

```

Below the code editor, the status bar indicates: "Result: query executed successfully. Took 1ms, 1 rows affected".

Deleting rows

Screenshot of SQL Server Management Studio showing a query window and a results window.

```
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
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
779
```

Result: query executed successfully. Took 0ms, 1 rows affected
At line 54:
UPDATE employees
SET lastname = 'Reddy'
WHERE

Delete all the rows and dropping a table

Screenshot of SQL Server Management Studio showing a query window and a results window.

```
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
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
```

Result: query executed successfully. Took 1ms, 270 rows affected
At line 579:
DELETE
FROM
 artists_backup;

Export table as csv (right click on table)

Screenshot of MySQL Workbench showing a tables browser and a results window.

Tables Browser:

- Tables (19)
 - OrderDetails
 - Products
 - PropertyInfo
 - SubTable
 - albums
 - artists
 - books
 - customers
 - employees
 - genres
 - invoices
 - media_types
 - playlist_track
 - playlists
 - products
 - queue
 - queue_sequence
 - queue_star
 - tracks
- Indices (10)
 - IFK_AlbumArtistId
 - IFK_AlbumGenreId
 - IFK_EmployeeReportTo
 - IFK_InvoiceLine
 - IFK_InvoiceLineInvoiceId
 - IFK_InvoiceLineTrackId
 - IFK_PlaylistTrackId
 - IFK_TrackAlbumId
 - IFK_TrackGenreId
 - IFK_TrackMediaTypeId
- Views (0)
- Triggers (0)

Results Window:

```
Edit Database Cell
Mode: Text
Text Import Export Set as Null
2013-01-01
Type of data currently in cells Text / Numeric
10 char(s)
Remote
Identity Commit Last modified Size
Name
```

Union

UNION is used to combine the result-set of two or more SELECT statements

Every SELECT statement within UNION must have the same number of columns

The columns must also have similar data types

The columns in every SELECT statement must also be in the same order

The screenshot shows a SQL editor window titled "Sample Work.sql" with the following code:

```
684 -- Union
685 ---UNION is used to combine the result-set of two or more SELECT statements
686 ---Every SELECT statement within UNION must have the same number of columns
687 ---The columns must also have similar data types
688 ---The columns in every SELECT statement must also be in the same order
689
690 SELECT City FROM customers
691 UNION
692 SELECT City FROM employees
693
694
695
696
697
698
699
```

Below the code, the results of the query are displayed in a table:

City
1 Amsterdam
2 Bangalore
3 Berlin
4 Bordeaux
5 Boston

Execution finished without errors.
Result: 55 rows returned in 2ms
At line 692:
SELECT City FROM customers
UNION

On the right side of the interface, there is a "NULL" value viewer and a "Remote" connection configuration panel.

Union All

UNION selects only distinct values by default. To allow duplicate values, use UNION ALL

The screenshot shows a SQL editor window titled "Sample Work.sql" with the following code:

```
688 ---Every SELECT statement within UNION must have the same number of columns
689 ---The columns must also have similar data types
690 ---The columns in every SELECT statement must also be in the same order
691
692 SELECT City FROM customers
693 UNION
694 SELECT City FROM employees
695
696
697 --- Union All
698 --- UNION selects only distinct values by default. To allow duplicate values, use UNION ALL
699
700
701 SELECT City FROM customers
702 UNION ALL
703 SELECT City FROM employees
```

Below the code, the results of the query are displayed in a table:

City
1 São José dos Campos
2 Stuttgart
3 Montréal
4 Oslo
5 Prague

Execution finished without errors.
Result: 67 rows returned in 2ms
At line 701:
SELECT City FROM customers
UNION ALL

On the right side of the interface, there is a "NULL" value viewer and a "Remote" connection configuration panel.

SQL Set Operators (Union, Union All, Minus, Intersect)

<https://learnsql.com/blog/introducing-sql-set-operators-union-union-minus-intersect/>

Stored Procedure

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So, if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

Demo Database

Below is a selection from the "Customers" table in the Northwind sample database:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Stored Procedure Example

The following SQL statement creates a stored procedure named "SelectAllCustomers" that selects all records from the "Customers" table:

Example

```
CREATE PROCEDURE SelectAllCustomers
AS
SELECT * FROM Customers
GO;
```

Execute the stored procedure above as follows:

Example

```
EXEC SelectAllCustomers;
```

Stored Procedure With One Parameter

The following SQL statement creates a stored procedure that selects Customers from a particular City from the "Customers" table:

Example

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)
AS
SELECT * FROM Customers WHERE City = @City
GO;
```

Execute the stored procedure above as follows:

Example

```
EXEC SelectAllCustomers @City = 'London';
```

Stored Procedure With Multiple Parameters

Setting up multiple parameters is very easy. Just list each parameter and the data type separated by a comma as shown below.

The following SQL statement creates a stored procedure that selects Customers from a particular City with a particular PostalCode from the "Customers" table:

Example

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode nvarchar(10)
AS
SELECT * FROM Customers WHERE City = @City AND PostalCode = @PostalCode
GO;
```

Execute the stored procedure above as follows:

Example

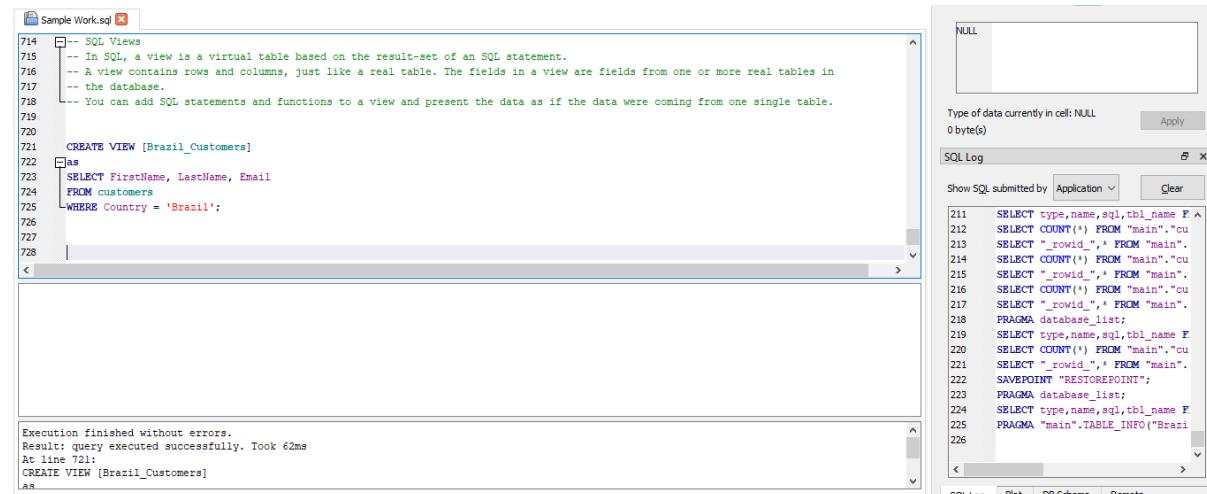
```
EXEC SelectAllCustomers @City = 'London', @PostalCode = 'WA1 1DP';
```

SQL Views

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.



```
Sample Work.sql
714 --- SQL Views
715 --- In SQL, a view is a virtual table based on the result-set of an SQL statement.
716 --- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in
717 --- the database.
718 --- You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.
719
720
721 CREATE VIEW [Brazil_Customers]
722 AS
723 SELECT FirstName, LastName, Email
724 FROM customers
725 WHERE Country = 'Brazil';
726
727
728
Execution finished without errors.
Result: query executed successfully. Took 62ms
At line 721:
CREATE VIEW [Brazil_Customers]
AS
```

Type of data currently in cell: NULL
0 byte(s)

```
SQL Log Application Clear
211 SELECT type,name,sql,tbl_name F ^
212 SELECT COUNT(*) FROM "main","cu
213 SELECT "_rowid_","^" FROM "main".
214 SELECT COUNT(*) FROM "main"."cu
215 SELECT "_rowid_","^" FROM "main".
216 SELECT COUNT(*) FROM "main"."cu
217 SELECT "_rowid_","^" FROM "main".
218 PRAGMA database_list;
219 SELECT type,name,sql,tbl_name F ^
220 SELECT COUNT(*) FROM "main","cu
221 SELECT "_rowid_","^" FROM "main".
222 SAVEPOINT "RESTOREPOINT";
223 PRAGMA database_list;
224 SELECT type,name,sql,tbl_name F ^
225 PRAGMA "main".TABLE_INFO("Brazil_Customers");
226
```

SQL Log Plot DB Schema Remote

We can query the view above as follows:

The screenshot shows the SQL Editor window with the following code:

```
717 -- the database.
718 -- You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.
719
720 CREATE VIEW [Brazil_Customers]
721   as
722     SELECT FirstName, LastName, Email
723       FROM customers
724      WHERE Country = 'Brazil';
725
726
727 -- We can query the view above as follows:
728
729   SELECT * FROM Brazil_Customers;
730
731
```

The results pane displays the following table:

FirstName	LastName	Email
1 Luís	Gonçalves	luisg@embraer.com.br
2 Eduardo	Martins	eduardo@woodstock.com.br
3 Alexandre	Rocha	aler0@uol.com.br
4 Roberto	Almeida	roberto.almeida@riotur.gov.br
5 Fernanda	Ramos	fernadaramos4@uol.com.br

Execution finished without errors.
Result: 5 rows returned in 3ms
At line 730:
SELECT * FROM Brazil_Customers;

On the right, the SQL Log pane shows the following log entries:

```
216   SELECT COUNT(*) FROM "main"."cu"
217   SELECT "_rowid_.*" FROM "main".
218 PRAGMA database_list;
219 SELECT type,name,sql,tbl_name F
220 SELECT COUNT(*) FROM "main"."cu"
221 SELECT "_rowid_.*" FROM "main".
222 SAVEPOINT "RESTOREPOINT";
223 PRAGMA database_list;
224 SELECT type,name,sql,tbl_name F
225 PRAGMA "main".TABLE_INFO("Braz
226 PRAGMA database_list;
227 SELECT type,name,sql,tbl_name F
228 PRAGMA "main".TABLE_INFO("Braz
229 SELECT COUNT(*) FROM (SELECT *
230   SELECT * FROM Brazil_Customers
231
```

View: Top 5 Customers by Sales

The screenshot shows the SQL Editor window with the following code:

```
735
736   CREATE VIEW [Top_5_Customers]
737   as
738     SELECT
739       a.CustomerId,
740       a.FirstName,
741       sum(c.UnitPrice * c.Quantity) as Total_Sales
742     FROM
743       customers a
744     INNER JOIN invoices b on a.CustomerId = b.CustomerId
745     INNER JOIN invoice_items c on b.InvoiceId = c.InvoiceId
746     GROUP by 1,2
747     ORDER by 3 DESC LIMIT 5;
748
749   SELECT * FROM Top_5_Customers;
750
```

The results pane displays the following table:

CustomerId	FirstName	Total_Sales
1	6 Helena	49.62
2	26 Richard	47.62
3	57 Luis	46.62
4	45 Ladislav	45.62
5	46 Hugh	45.62

Execution finished without errors.
Result: 5 rows returned in 93ms
At line 749:
SELECT * FROM Top_5_Customers;

On the right, the Remote pane shows the connection status:

```
Identity: Select an identity to connect
DBHub.io Local Current Database
Name:
```

SQL Set Operators (Union, UnionAll, Minus, Intersect)

<https://learnsql.com/blog/introducing-sql-set-operators-union-union-minus-intersect/>

SQL Practice Exercises

<https://www.w3resource.com/sql-exercises/sql-joins-exercises.php>

<https://www.w3resource.com/sql-exercises/joins-hr/index.php>

Types of Indexes (Clustered vs Non-Clustered Indexes)

<https://www.spotlightcloud.io/blog/when-to-use-clustered-or-non-clustered-indexes-in-sql-server>

Types of Views (Materialised vs Non-Materialised)

<https://www.besanttechnologies.com/difference-between-view-and-materialized-view>

Replace Values in a Column

Update [Table Name]

Set [Column Name] = New Value

where [Column Name] = Old Value;

Delete Rows with Null values

Delete from [Table Name]

where [Column Name] is null;

Replace Column Name in a Table

Alter Table [Table Name]

Rename Column Old Column Name To New Column Name

Transaction Query Language (TQL) / Transaction Control Language (TCL)

<https://sebhastian.com/mysql-transaction/>

<https://www.mysqltutorial.org/mysql-transaction.aspx>

4. **TCL(Transaction Control Language)** : TCL commands deals with the transaction within the database.

Examples of TCL commands:

- **COMMIT**– commits a Transaction.
- **ROLLBACK**– rollbacks a transaction in case of any error occurs.
- **SAVEPOINT**–sets a savepoint within a transaction.
- **SET TRANSACTION**–specify characteristics for the transaction.

Triggers

<https://www.geeksforgeeks.org/sql-trigger-student-database/>

<https://learn.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver16>

https://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm

Common Table Expressions (CTE)

<https://www.geeksforgeeks.org/cte-in-sql/>

<https://www.sqlshack.com/sql-server-common-table-expressions-cte/>