

# Outer Joins

# Outer Joins

```
SELECT * FROM Author;
```

aid	name
1	ab
2	cd
3	ef

```
SELECT * FROM Book;
```

bid	aid
1	1
2	1
3	3

```
SELECT * FROM Book JOIN Author USING (aid);
```

aid	bid	name
1	1	ab
1	2	ab
3	3	ef

Author "2" not listed,  
because he/she not in the Book-table.

# Outer Joins

```
SELECT * FROM Author;
```

aid	name
1	ab
2	cd
3	ef

```
SELECT * FROM Book;
```

bid	aid
1	1
2	1
3	3

```
SELECT * FROM Book JOIN Author USING (aid);
```

aid	bid	name
1	1	ab
1	2	ab
3	3	ef

```
SELECT * FROM Book RIGHT OUTER JOIN  
Author USING (aid);
```

aid	name	bid
1	ab	1
1	ab	2
2	cd	NULL
3	ef	3

# Outer Joins

```
SELECT * FROM Author;
```

aid	name
1	ab
2	cd
3	ef

```
SELECT * FROM Book;
```

bid	aid
1	1
2	1
3	3

```
SELECT * FROM Book RIGHT OUTER JOIN  
Author USING (aid);
```

aid	name	bid
1	ab	1
1	ab	2
2	cd	NULL
3	ef	3

```
SELECT aid, count (bid) AS n_books  
FROM Book RIGHT OUTER JOIN  
Author USING (aid)  
GROUP BY aid;
```

aid	n_books
1	2
2	0
3	1



Here, NULL means “there is no value”, instead of “unknown value”

# Outer Joins

```
Table1 RIGHT OUTER JOIN Table2 USING / ON ...
```

- joins all tuples of **Table1** with **Table2** satisfying join condition, plus all remaining tuples from **Table2** (the RIGHT)
- result tuples of the second type above have **NULL-values** in the columns coming from **Table1**.

# Outer Joins



```
Table1 RIGHT OUTER JOIN Table2 USING / ON ...
```

- joins all tuples of **Table1** with **Table2** satisfying join condition, plus all remaining tuples from **Table2** (the RIGHT)
- result tuples of the second type above have **NULL-values** in the columns coming from **Table1**.



```
Table1 LEFT OUTER JOIN Table2 USING / ON ...
```

- joins all tuples of **Table1** with **Table2** satisfying join condition, plus all remaining tuples from **Table1** (the LEFT)
- result tuples of the second type above have **NULL-values** in the columns coming from **Table2**.

Join on all  
common attributes

```
SELECT * FROM Part;
```

part_id	supp_id
P1	S1
P2	S2
P3	NULL
P4	NULL

```
SELECT * from Supplier;
```

supp_id	supp_name
S1	Supplier#1
S2	Supplier#2
S3	Supplier#3

```
SELECT * FROM Part NATURAL JOIN Supplier;
```

supp_id	part_id	supp_name
S1	P1	Supplier#1
S2	P2	Supplier#2

# Left Outer Join

```
SELECT * FROM Part;
```

part_id	supp_id
P1	S1
P2	S2
P3	NULL
P4	NULL

```
SELECT * from Supplier;
```

supp_id	supp_name
S1	Supplier#1
S2	Supplier#2
S3	Supplier#3

```
SELECT part_id, supp_name FROM  
Part NATURAL LEFT JOIN Supplier;
```

part_id	supp_name
P1	Supplier#1
P2	Supplier#2
P3	NULL
P4	NULL



# Right Outer Join

```
SELECT * FROM Part;
```

part_id	supp_id
P1	S1
P2	S2
P3	NULL
P4	NULL

```
SELECT * from Supplier;
```

supp_id	supp_name
S1	Supplier#1
S2	Supplier#2
S3	Supplier#3

```
SELECT part_id, supp_name FROM
```

```
Part NATURAL LEFT JOIN Supplier;
```

part_id	supp_name
P1	Supplier#1
P2	Supplier#2
P3	NULL
P4	NULL

```
SELECT part_id, supp_name FROM
```

```
Part NATURAL RIGHT JOIN Supplier;
```

part_id	supp_name
P1	Supplier#1
P2	Supplier#2
NULL	Supplier#3

# Full Outer Join

```
SELECT * FROM Part;
```

part_id	supp_id
P1	S1
P2	S2
P3	NULL
P4	NULL

```
SELECT * from Supplier;
```

supp_id	supp_name
S1	Supplier#1
S2	Supplier#2
S3	Supplier#3

```
Part NATURAL LEFT JOIN Supplier;
```

part_id	supp_name
P1	Supplier#1
P2	Supplier#2
P3	NULL
P4	NULL

```
Part NATURAL RIGHT JOIN Supplier;
```

part_id	supp_name
P1	Supplier#1
P2	Supplier#2
NULL	Supplier#3

```
Part NATURAL FULL OUTER JOIN Supplier;
```

part_id	supp_name
P1	Supplier#1
P2	Supplier#2
P3	NULL
P4	NULL
NULL	Supplier#3

# Full Outer Join

```
SELECT * FROM Part;
```

part_id	supp_id
P1	S1
P2	S2
P3	NULL
P4	NULL

```
SELECT * from Supplier;
```

supp_id	supp_name
S1	Supplier#1
S2	Supplier#2
S3	Supplier#3

— **sqlite3** does not support outer joins.

— how to simulate them?  
(will be done in the Exercises)

```
Part NATURAL LEFT JOIN Supplier;
```

part_id	supp_name
P1	Supplier#1
P2	Supplier#2
P3	NULL
P4	NULL

```
Part NATURAL RIGHT JOIN Supplier;
```

part_id	supp_name
P1	Supplier#1
P2	Supplier#2
NULL	Supplier#3

```
Part NATURAL FULL OUTER JOIN Supplier;
```


part_id	supp_name
P1	Supplier#1
P2	Supplier#2
P3	NULL
P4	NULL
NULL	Supplier#3

# Outer Join

Outer joins can be quite useful:


Ex1	name	exam1
	student-1	85
	student-3	90
	student-4	68
	student-9	77

no  
student-5



Ex2	name	exam2
	student-1	87
	student-3	92
	student-5	90
	student-9	80

no  
student-4



Ex3	name	exam3
	student-1	85
	student-3	90
	student-4	68
	student-5	60
	student-9	77


The **NATURAL JOIN** of these tables would  
**remove** student-4 and student-5!

# Outer Join

Outer joins can be quite useful:


Ex1	name	exam1
	student-1	85
	student-3	90
	student-4	68
	student-9	77

no  
student-5



Ex2	name	exam2
	student-1	87
	student-3	92
	student-5	90
	student-9	80

no  
student-4



Ex3	name	exam3
	student-1	85
	student-3	90
	student-4	68
	student-5	60
	student-9	77

The **NATURAL JOIN** of these tables would  
**remove** student-4 and student-5!

**Instead:** use (twice) a **NATURAL FULL OUTER JOIN**!

## Cleaning up the movies.sqlite3 database

1.) which columns contain EMPTY strings?

```
sqlite> .tables
actors2awards          movies2grossopeningweekend
actors2movies          movies2grossworldwide
awards                 persons
awards2movies          producers2awards
directors2awards       producers2movies
directors2movies       ratings
genres                 runtimes
locations              writers2awards
movies                 writers2movies
movies2budget
sqlite> .schema actors2awards
CREATE TABLE actors2awards(personid text REFERENCES persons(personid),
                             awardid text REFERENCES awards(awardid),
                             PRIMARY KEY (personid,awardid));
sqlite> select count(*) from actors2awards where personid="";
0
sqlite> select count(*) from actors2awards where awardid="";
0
sqlite>
```

==> no EMPTY strings in actors2awards

```
sqlite> .tables
actors2awards          movies2grossopeningweekend
actors2movies          movies2grossworldwide
awards                 persons
awards2movies          producers2awards
directors2awards       producers2movies
directors2movies       ratings
genres                 runtimes
locations              writers2awards
movies                 writers2movies
movies2budget
sqlite> .schema actors2movies
CREATE TABLE actors2movies(personid text REFERENCES persons(personid),
                             movieid text REFERENCES movies(movieid),
                             ascharacter text,
                             PRIMARY KEY (movieid,personid));
sqlite> select count(*) from actors2movies;
1048575
sqlite> select count(*) from actors2movies where ascharacter is null;
0
sqlite> select count(*) from actors2movies where ascharacter="";
36865
sqlite>
```

==> it should NOT be the case that (movieid,personid) is the PRIMARY KEY.

==> there are movies where an actors plays more than one character,  
but unfortunately our database does not have this information.



```
CREATE TABLE actors2movies(personid text REFERENCES persons(personid),
                             movieid text REFERENCES movies(movieid),
                             ascharacter text,
                             PRIMARY KEY (movieid, personid));

sqlite> select count(*) from actors2movies;
1048575
sqlite> select count(*) from actors2movies where ascharacter is null;
0
sqlite> select count(*) from actors2movies where ascharacter="";
36865
sqlite>
```

Let's leave this table as it is.

==> I will try to fix this table by getting the missing entries on ischarater-information.

==> it should NOT be the case that (movieid, personid) is the PRIMARY KEY.

==> there are movies where an actors plays more than one character, but unfortunately our database does not have this information.

```
genres          runtimes
locations       writers2awards
movies          writers2movies
movies2budget

sqlite> .schema awards
CREATE TABLE awards(awardid text PRIMARY KEY,
                    awardtitle text NOT NULL,
                    awardyear text NOT NULL,
                    awardcategory text NOT NULL,
                    awardoutcome text NOT NULL,
                    awarddescription text);

sqlite> select count(*) from awards where awardtitle="";
0
sqlite> select count(*) from awards where awardyear="";
0
sqlite> select count(*) from awards where awardcategory="";
0
sqlite> select count(*) from awards where awardoutcome="";
0
sqlite> select count(*) from awards where awarddescription="";
Error: in prepare, no such column: awarddescription (1)
sqlite> select count(*) from awards where awarddescription="";
1967
sqlite>
```

create.sql = CREATE command for each table

```
CREATE TABLE awards(awardid text PRIMARY KEY,  
                    awardtitle text NOT NULL,  
                    awardyear text NOT NULL,  
                    awardcategory text NOT NULL,  
                    awardoutcome text NOT NULL,  
                    awarddescription text);
```

Let's first fix the typo

Now:

```
ALTER TABLE awards RENAME TO awards_original;
```

File Edit Options Buffers Tools SQL Help

```
ALTER TABLE awards RENAME TO awards_original;
```

```
CREATE TABLE awards(awardid text PRIMARY KEY,  
                      awardtitle text NOT NULL,  
                      awardyear text NOT NULL,  
                      awardcategory text NOT NULL,  
                      awardoutcome text NOT NULL);
```

```
insert into awards select awardid,awardtitle,awardyear,awardcategory,awardoutcome  
from awards_original;
```

```
create table awards2description(awardid text primary key,  
                                awarddescription text not null);
```

```
insert into awards2description select awardid,awarddescription  
from awards_original where awarddescription!="";
```

-UU-:\*\*--F1 create.sql Bot L107 (SQL[ANSI]) -----

“Saneness” check:

```
smaneth — screen -R — 87x24
sqlite> select count(*) from awards2description;
266592
sqlite> select count(*) from awards_original ;
268559
sqlite> select 268559-266592;
1967
sqlite>
```

“Saneness” check:

- how can you do this in a much better way?
- Idea: **LEFT NATURAL JOIN** of the new tables should be **the same** as the original table!
- Check if two tables (both without duplicates and with the same schema) are the same:

```
SELECT * from table1 EXCEPT SELECT * from table2;
```

==> should give an empty results

```
SELECT * from table2 EXCEPT SELECT * from table1;
```

==> should give an empty results

---

For us we would first need to change EMTPY string to NULL  
or NULL to EMPTY string.....

Find employees that are not in the sales department table

## Tables

Employees Table:

EmployeeID	EmployeeName	Department
1	John	Sales
2	Sarah	HR
3	Mark	IT
4	Lisa	Finance
5	Alex	Sales

SalesEmployees Table:

EmployeeID
1
5

# 'NOT IN'

## Tables

Employees Table:


EmployeeID	EmployeeName	Department
1	John	Sales
2	Sarah	HR
3	Mark	IT
4	Lisa	Finance
5	Alex	Sales

SalesEmployees Table:

EmployeeID
1
5

## QUERY

sql

 Copy code

```
SELECT EmployeeID, EmployeeName
FROM Employees
WHERE EmployeeID NOT IN (SELECT EmployeeID FROM SalesEmployees);
```

## RESULT

EmployeeID	EmployeeName
2	Sarah
3	Mark
4	Lisa



# 'NOT EXISTS'

## Tables

Employees Table:

EmployeeID	EmployeeName	Department
1	John	Sales
2	Sarah	HR
3	Mark	IT
4	Lisa	Finance
5	Alex	Sales

SalesEmployees Table:

EmployeeID
1
5

## QUERY

```
SELECT EmployeeID, EmployeeName
FROM Employees
WHERE NOT EXISTS
(SELECT EmployeeID FROM SalesEmployees
WHERE SalesEmployees.EmployeeID = Employees.EmployeeID);
```

## RESULT

EmployeeID	EmployeeName
2	Sarah
3	Mark
4	Lisa

# How can we use this in our database

- Find persons that are not producers using 'not in' and 'not exist'

# How can we use this in our database

- Find persons that are not producers using 'not in' and 'not exists'

```
sqlite> select personid, name from persons where personid not in (select personid from producers2movies)
limit 5;
+-----+-----+
| personid |      name      |
+-----+-----+
| nm0660139 | Salvatore Papa |
| nm0209738 | Giuseppe de Liguoro |
| nm1375863 | Augusto Milla  |
| nm3942815 | Pier Delle Vigne |
| nm1374534 | Emilise Beretta |
+-----+-----+
sqlite> select personid, name from persons where not exists (select 1 from directors2movies where directors2movies.personid = persons.personid) limit 5;
+-----+-----+
| personid |      name      |
+-----+-----+
| nm0660139 | Salvatore Papa |
| nm1375863 | Augusto Milla  |
| nm3942815 | Pier Delle Vigne |
| nm1374534 | Emilise Beretta |
| nm0685283 | Arturo Pirovano |
+-----+-----+
sqlite> □
```

# EXCEPT in dbms

- EXCEPT is used to compare rows returned by Table A with rows returned from Table B and return distinct rows from TableA that do not appear in Table B
- 'NOT in' , 'NOT EXISTS' and 'EXCEPT' have similar use cases but the syntax is different.

TableA:

Column1	Column2	Column3
1	A	X
2	B	Y
3	C	Z

TableB:

Column1	Column2	Column3
2	B	Y
4	D	W

```
sql Copy code  
  
SELECT Column1, Column2, Column3  
FROM TableA  
EXCEPT  
SELECT Column1, Column2, Column3  
FROM TableB;
```

The result of this query will be:

Column1	Column2	Column3
1	A	X
3	C	Z

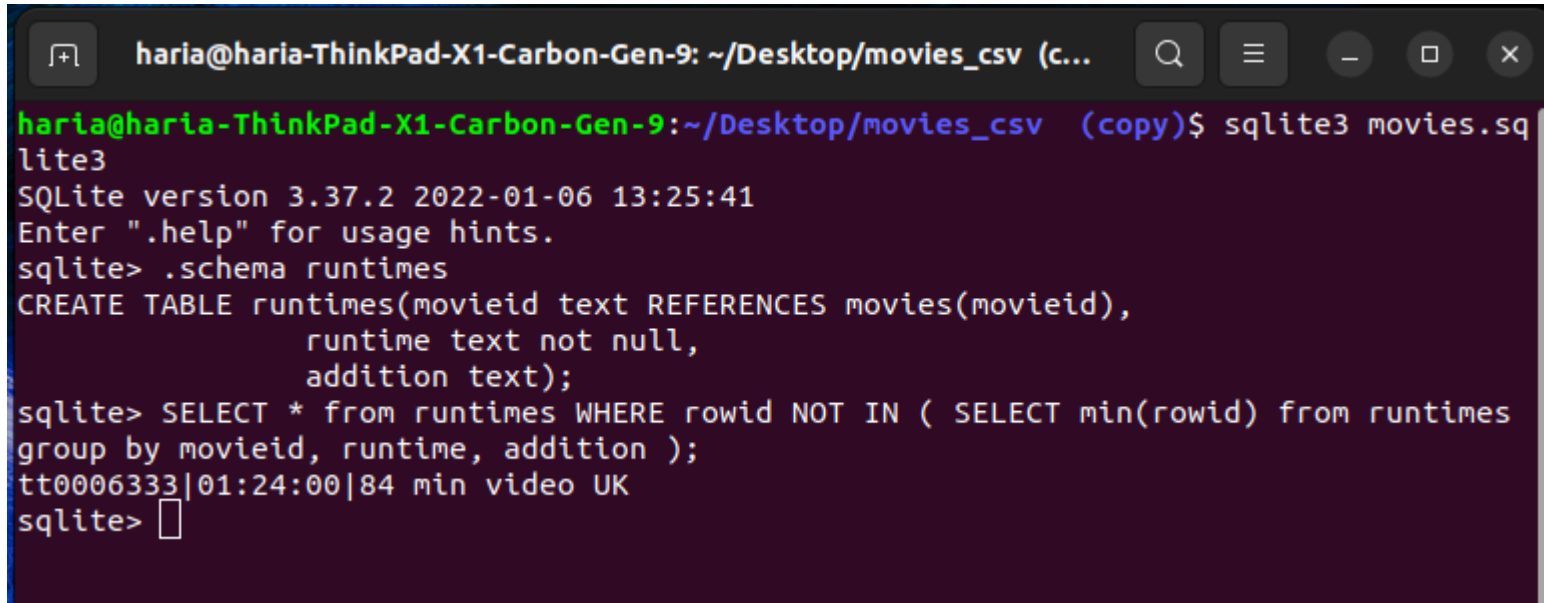
# How to check if a table has duplicates in sqlite3

- How to find the duplicates
- How to delete them

# How to check if a table has duplicates in sqlite3

Check if there are any duplicate rows in 'runtimes' table

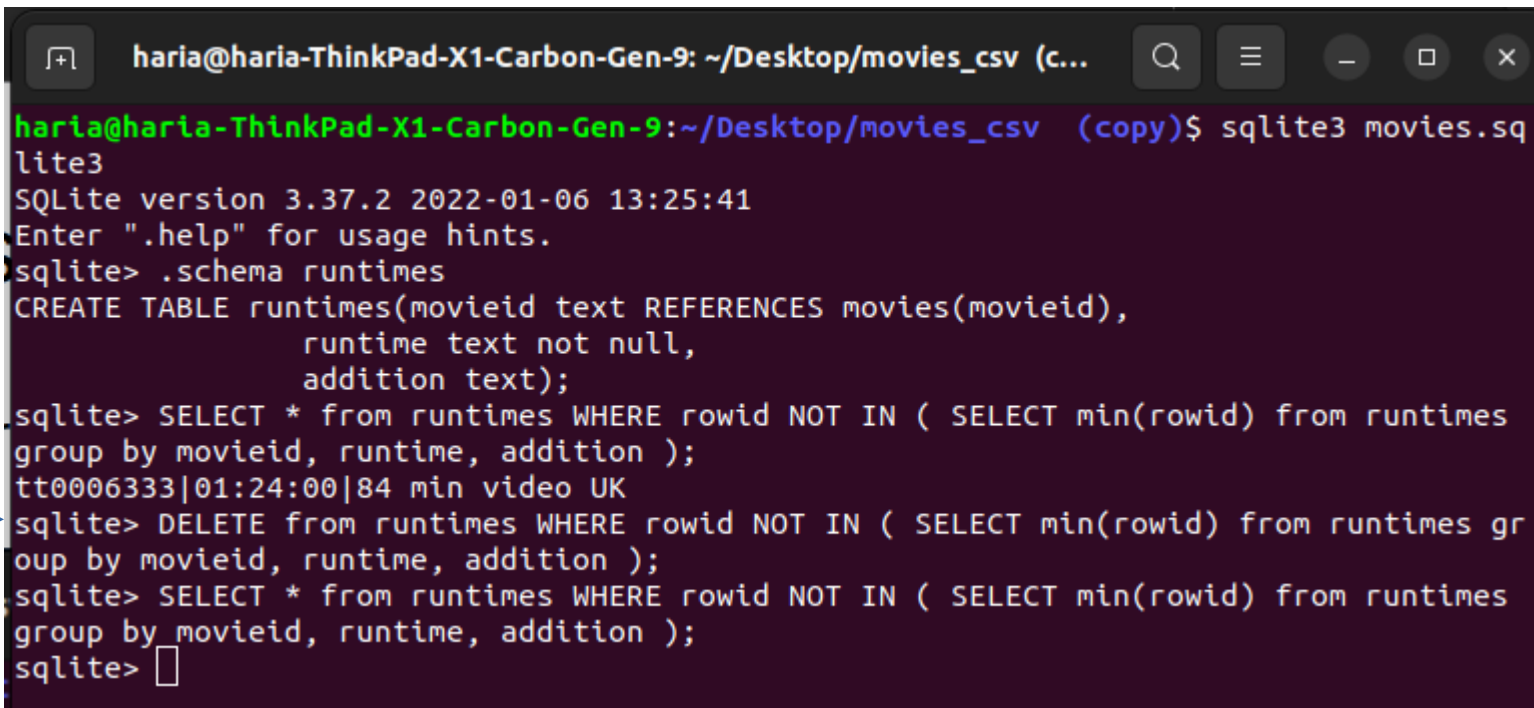
- RowID is a unique identifier assigned to each row in sqlite3



```
haria@haria-ThinkPad-X1-Carbon-Gen-9: ~/Desktop/movies_csv (c...
haria@haria-ThinkPad-X1-Carbon-Gen-9:~/Desktop/movies_csv (copy)$ sqlite3 movies.sq
lite3
SQLite version 3.37.2 2022-01-06 13:25:41
Enter ".help" for usage hints.
sqlite> .schema runtimes
CREATE TABLE runtimes(movieid text REFERENCES movies(movieid),
                        runtime text not null,
                        addition text);
sqlite> SELECT * from runtimes WHERE rowid NOT IN ( SELECT min(rowid) from runtimes
group by movieid, runtime, addition );
tt0006333|01:24:00|84 min video UK
sqlite>
```

# How to check if a table has duplicates in sqlite3

DELETE duplicate rows in 'runtimes' table and check again to confirm



```
haria@haria-ThinkPad-X1-Carbon-Gen-9: ~/Desktop/movies_csv (c...
haria@haria-ThinkPad-X1-Carbon-Gen-9:~/Desktop/movies_csv (copy)$ sqlite3 movies.sqlite3
SQLite version 3.37.2 2022-01-06 13:25:41
Enter ".help" for usage hints.
sqlite> .schema runtimes
CREATE TABLE runtimes(movieid text REFERENCES movies(movieid),
                      runtime text not null,
                      addition text);
sqlite> SELECT * from runtimes WHERE rowid NOT IN ( SELECT min(rowid) from runtimes
group by movieid, runtime, addition );
tt0006333|01:24:00|84 min video UK
sqlite> DELETE from runtimes WHERE rowid NOT IN ( SELECT min(rowid) from runtimes gr
oup by movieid, runtime, addition );
sqlite> SELECT * from runtimes WHERE rowid NOT IN ( SELECT min(rowid) from runtimes
group by movieid, runtime, addition );
sqlite>
```

# How to check if a table has duplicates in sqlite3

Check for duplicates in other tables

- Locations, directors2awards, writers2awards, producer2awards.

Find the duplicates and delete them

- I will upload the new movies.sqlite3 file without duplicates so that everyone has a clean database to work on.