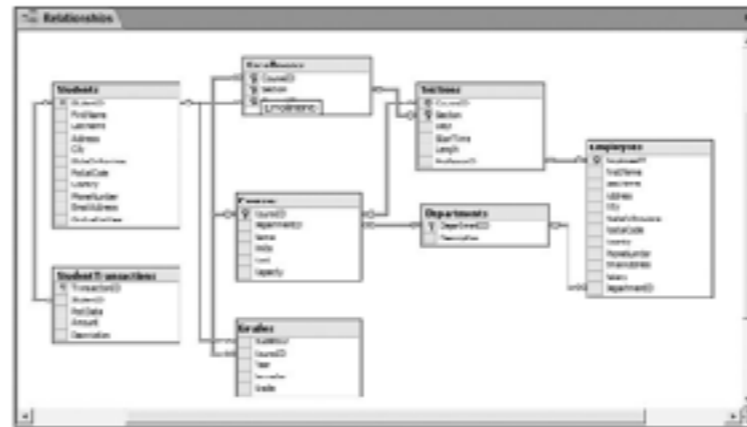


SQLite in Android

What is a database?

- **relational database:** A method of structuring data as **tables** associated to each other by shared attributes.
- a table **row** corresponds to a unit of data called a record; a **column** corresponds to an attribute of that record
- relational databases typically use Structured Query Language (**SQL**) to define, manage, and search data



Why use a database?

- **powerful:** can search, filter, combine data from many sources
- **fast:** can search/filter a database very quickly compared to a file
- **big:** scale well up to very large data sizes
- **safe:** built-in mechanisms for failure recovery (transactions)
- **multi-user:** concurrency features let many users view/edit data at same time
- **abstract:** layer of abstraction between stored data and app(s)
common syntax: database programs use same SQL commands

Relational database

- A database is a set of **tables**
 - Each table has a **primary key** — a column with unique values to identify a row
 - Tables can be related via **foreign keys**.

id	name	email
123	Bart	bart@fox.com
456	Milhouse	milhouse@fox.com
888	Lisa	lisa@fox.com
404	Ralph	ralph@fox.com

students

id	name
1234	Krabappel
5678	Hoover
9012	Stepp

teachers

id	name	teacher_id
10001	Computer Science 142	1234
10002	Computer Science 143	5678
10003	Computer Science 190M	9012
10004	Informatics 100	1234

courses

student_id	course_id	grade
123	10001	B-
123	10002	C
456	10001	B+
888	10002	A+
888	10003	A+
404	10004	D+

grades

Some database software

- **Oracle**
- Microsoft
 - **SQLServer**(powerful)
 - **Access**(simple)
- **PostgreSQL**
 - powerful/complex free open-source database system
- **SQLite**
 - transportable, lightweight free open-source database system
- **MySQL**
 - simple free open-source database system
 - many servers run “LAMP” (Linux,Apache,MySQL,andPHP)
 - Wikipedia is run on PHP and MySQL

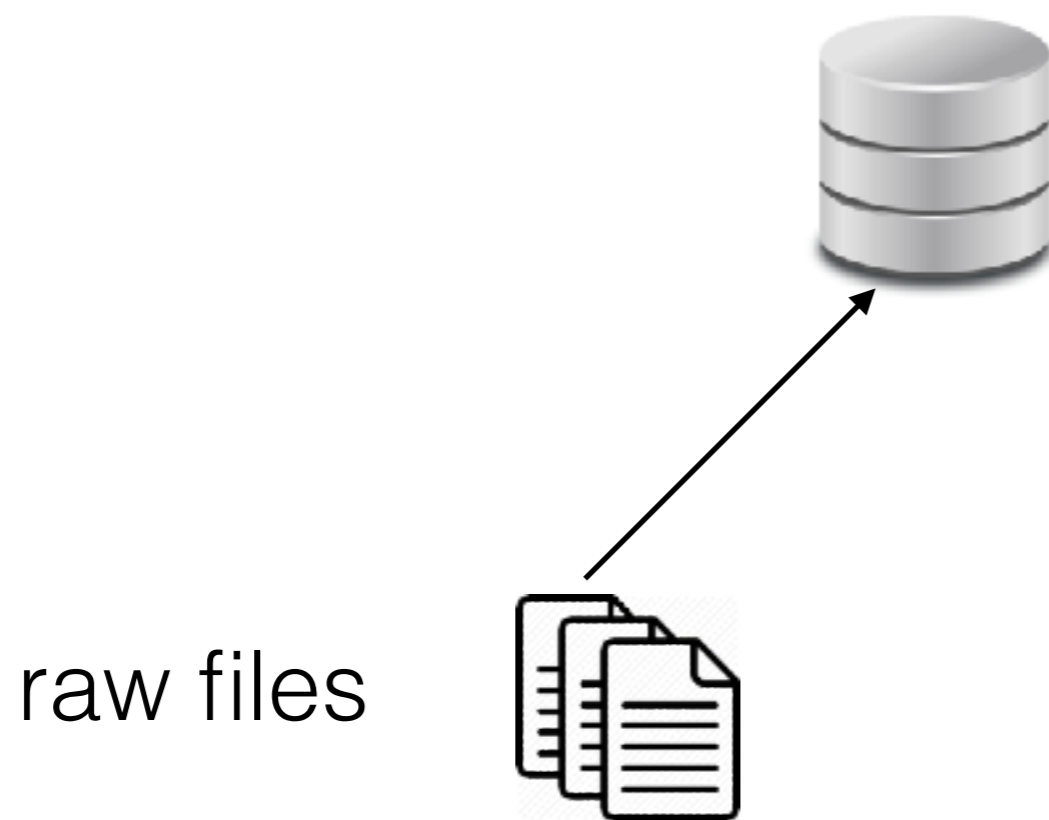
Android includes SQLite



SQLite is a library,
runs in the app's process

Android Media Manager (Media Content Provider)

- The Media provider contains meta data for all available media on both internal and external storage devices.



SQLite:

metadata:

- file location
- size
- artist
- albums
- playlists
- ...

The main table in Media: files

A single table to represent all types of media files:
Each row can be an image, audio, video, or playlist

_id	_data	_size	title	...
1	a.jpg	10000	a	
2	b.bmp	20000	b	
3	c.mp3	320000	c	
4	d.avi	12312000	d	

http://androidxref.com/4.4.3_r1.1/xref/packages/providers/MediaProvider/src/com/android/providers/media/MediaProvider.java#1335

Other tables in Media

- thumbnails,
- artists,
- albums,
- audio_playlists_map (stores members of a playlist)

Rows: Fixed number of columns

Tables: Variable number of rows

SQL

- **Structured Query Language (SQL):** a language for searching and updating a database
 - a standard syntax that is used by all database software (*with minor incompatibilities*)
 - generally case-insensitive
- a **declarative language:** describes what data you are seeking, not exactly how to find it

Basic SQL operations

- SELECT
- INSERT
- UPDATE
- DELETE

SELECT

- **SELECT** <list of columns> **FROM** <table>
WHERE <where clause>
[**ORDER BY** <column> [ASC or DESC]]
[**LIMIT** <number>];
- e.g., `SELECT * FROM files WHERE _id=3;`

_id	_data	_size	title	...
1	a.jpg	10000	a	
2	b.bmp	20000	b	
3	c.mp3	320000	c	
4	d.avi	12312000	d	

SELECT

- **SELECT** <list of columns> **FROM** <table>
WHERE <where clause>
[**ORDER BY** <column> [ASC or DESC]]
[**LIMIT** <number>];
 - SELECT _id, _data FROM files
 - SELECT * FROM files; (* means all columns)
- **ORDER BY**: sort the result by a column
- **LIMIT**: only get the first n rows in the result

INSERT

- **INSERT INTO** <table> (<list of columns>)
VALUES (<list of values>);
- e.g., **INSERT INTO** files (data, size, title)
VALUES ("image0.jpg", 102400, "image0");

_id	_data	_size	title	...
1	a.jpg	10000	a	
2	b.bmp	20000	b	
3	c.mp3	320000	c	
4	d.avi	12312000	d	
5	image0.jpg	102400	image0	

UPDATE

- **UPDATE** <table> **SET**
 <column1> = <value1>,
 <column2> = <value2>,
 ...
 <columnn> = <valuen>
WHERE <where clause>;

UPDATE

- e.g., **UPDATE** files **SET** title="profile"
WHERE _id=5;

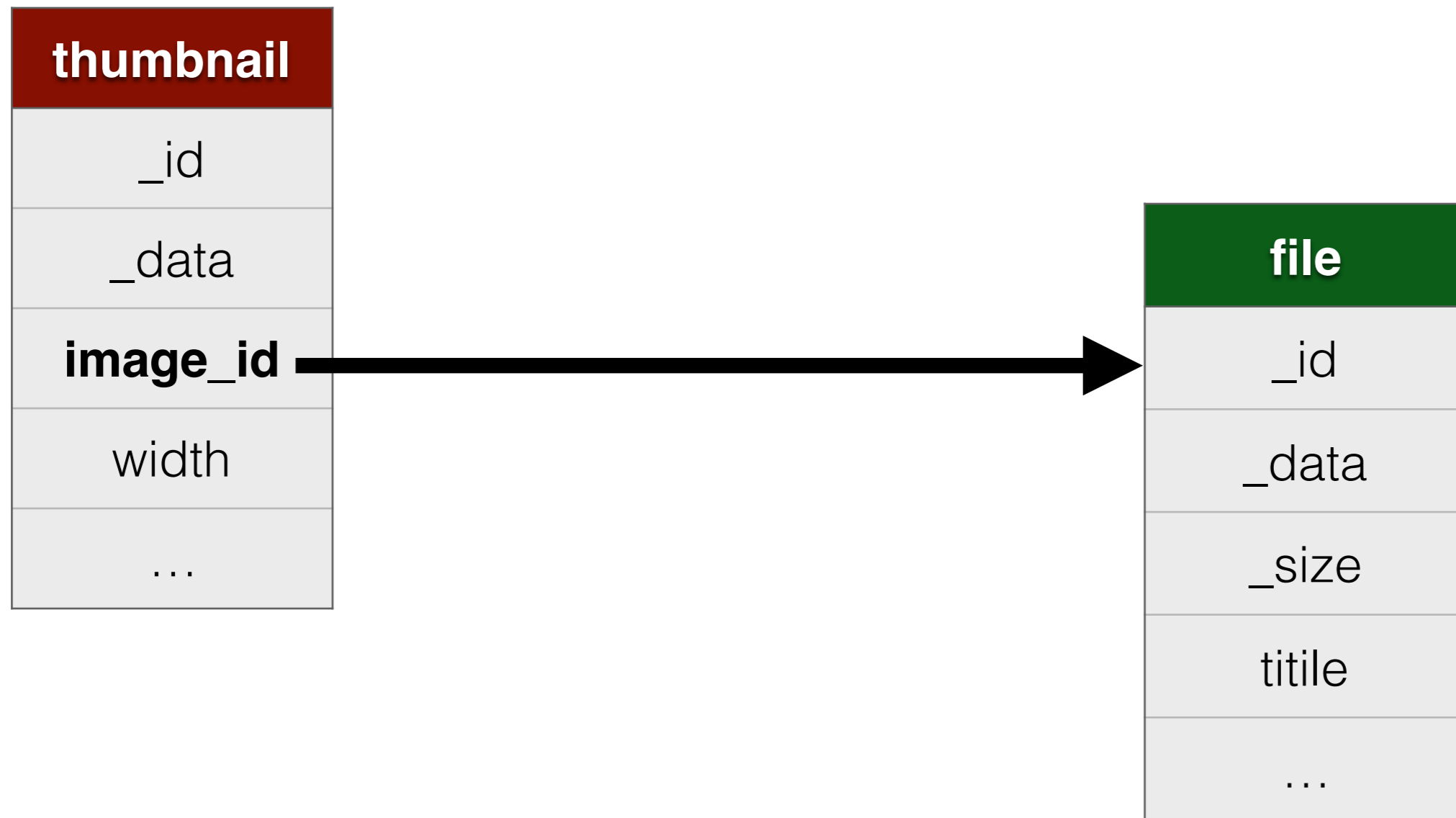
_id	_data	_size	title	...
1	a.jpg	10000	a	
2	b.bmp	20000	b	
3	c.mp3	320000	c	
4	d.avi	12312000	d	
5	image0.jpg	102400	profile	

DELETE

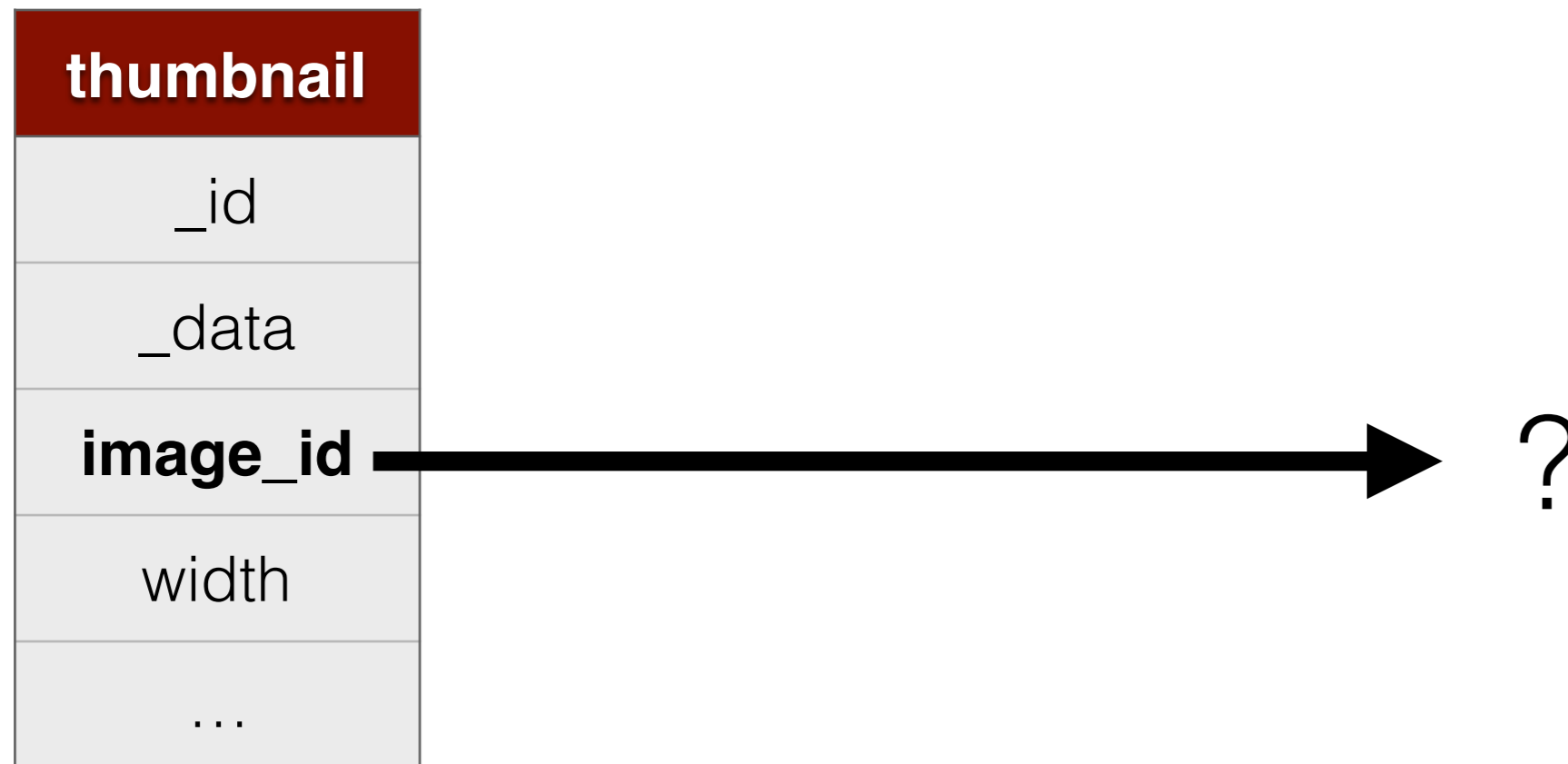
- **DELETE FROM** <table>
WHERE <where clause>;
- e.g., **DELETE FROM** files
WHERE _id=4;

_id	_data	_size	title	...
1	a.jpg	10000	a	
2	b.bmp	20000	b	
3	c.mp3	320000	c	
4	d.avi	12312000	d	
5	image0.jpg	102400	profile	

Related data across tables



Related data across tables



Foreign keys

If **thumbnails.image_id** is declared to be a ***foreign key*** of **files._id**,

SQLite will enforce ***Referential Integrity***:

When a row in files is removed or its _id is changed, SQLite can set the affected foreign keys in thumbnails to NULL, or remove the affected rows, etc.

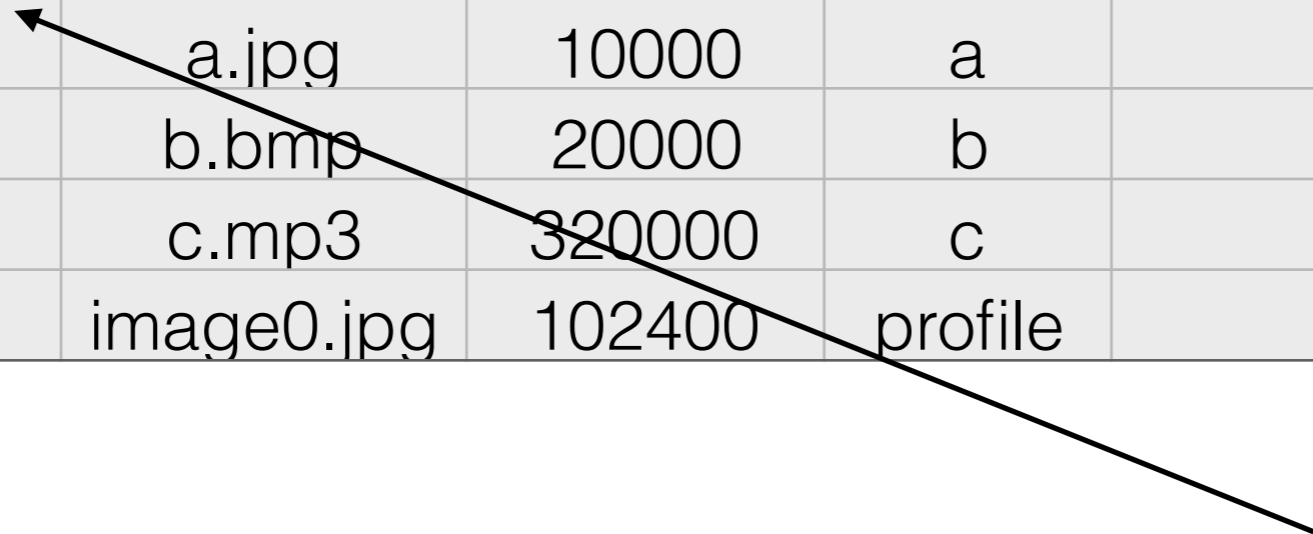
Foreign keys

files table

_id	_data	_size	title	...
1	a.jpg	10000	a	
2	b.bmp	20000	b	
3	c.mp3	320000	c	
5	image0.jpg	102400	profile	

thumbnails table

_id	_data	image_id	width	...
1	1.thumb	1	300	
3	5.thumb	5	600	



ON DELETE CASCADE

files table

_id	_data	_size	title	...
1	a.jpg	10000	a	
2	b.bmp	20000	b	
3	c.mp3	320000	c	
5	image0.jpg	102400	profile	

thumbnails table

_id	_data	image_id	width	...
1	1.thumb	1	300	
3	5.thumb	5	600	

ON DELETE SET NULL

files table

_id	_data	_size	title	...
1	a.jpg	10000	a	
2	b.bmp	20000	b	
3	c.mp3	320000	c	
5	image0.jpg	102400	profile	

thumbnails table

_id	_data	image_id	width	...
1	1.thumb	NULL	300	
3	5.thumb	5	600	

Join — query multiple related tables

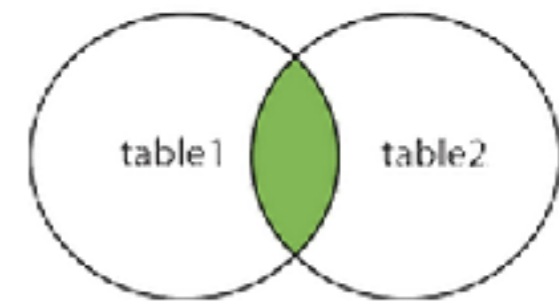
- Inner join
- Outer join

If multiple tables have the same column name, use `<table>.<col>` to distinguish them

Inner Join

- Inner join (JOIN) — only returns rows matching the condition

- **SELECT ... FROM** files
JOIN thumbnails
ON files._id=thumbnails.image_id
WHERE ...



- Equivalent to

- **SELECT ... FROM** files, thumbnails
WHERE files._id=thumbnails.image_id
AND (...)

Inner Join

files

_id	_data	_size	title	...
1	a.jpg	10000	a	
2	b.bmp	20000	b	
5	image0.jpg	102400	profile	

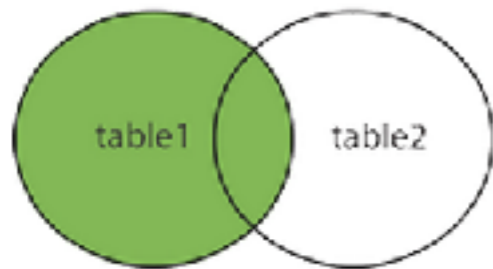
thumbnails

_id	_data	image_id	width	...
1	1.thumb	1	300	
3	5.thumb	5	600	

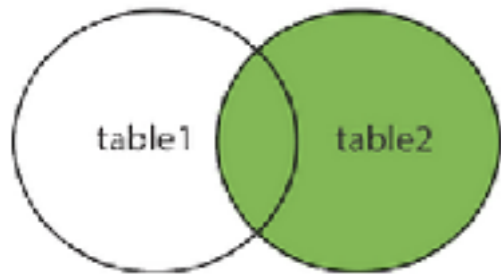
JOIN ON files._id=thumbnails.image_id

files._id	title	...	thumbnails._id	width	...
1	a		1	300	
5	profile		3	600	

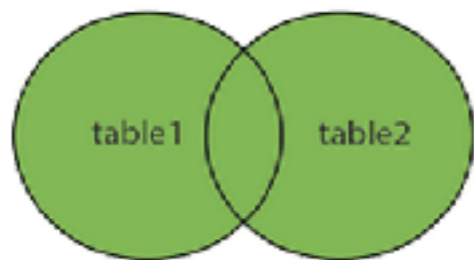
Outer Join



Left outer join (LEFT [OUTER] JOIN) — returns all rows in the left table, fill NULL to the right table if no matching rows.



Right outer join — returns all rows in the right table, fill NULL to the left table if no matching rows. (not supported by SQLite)



Full outer join — records from both sides are included, fill NULL to “the other table” if no match. (not supported by SQLite)

Left Outer Join

- Left outer join (LEFT [OUTER] JOIN) — returns all rows in the left table, fill NULL to the right table if no matching rows.
- **SELECT ... FROM** files
LEFT OUTER JOIN thumbnails
ON files._id=thumbnails.image_id
WHERE ...

Left Outer Join

files

_id	_data	_size	title	...
1	a.jpg	10000	a	
2	b.bmp	20000	b	
5	image0.jpg	102400	profile	

thumbnails

_id	_data	image_id	width	...
1	1.thumb	1	300	
3	5.thumb	5	600	

JOIN ON files._id=thumbnails.image_id


files._id	title	...	thumbnails._id	width	...
1	a		1	300	
2	b		NULL	NULL	
5	profile		3	600	

Views

A view is a virtual table based on other tables or views

CREATE VIEW <view name> **AS**
SELECT

_id	_data	_size	title	type	...
1	a.jpg	10000	a	image	
2	b.bmp	20000	b	image	
3	c.mp3	320000	c	audio	
5	image0.jpg	102400	profile	image	

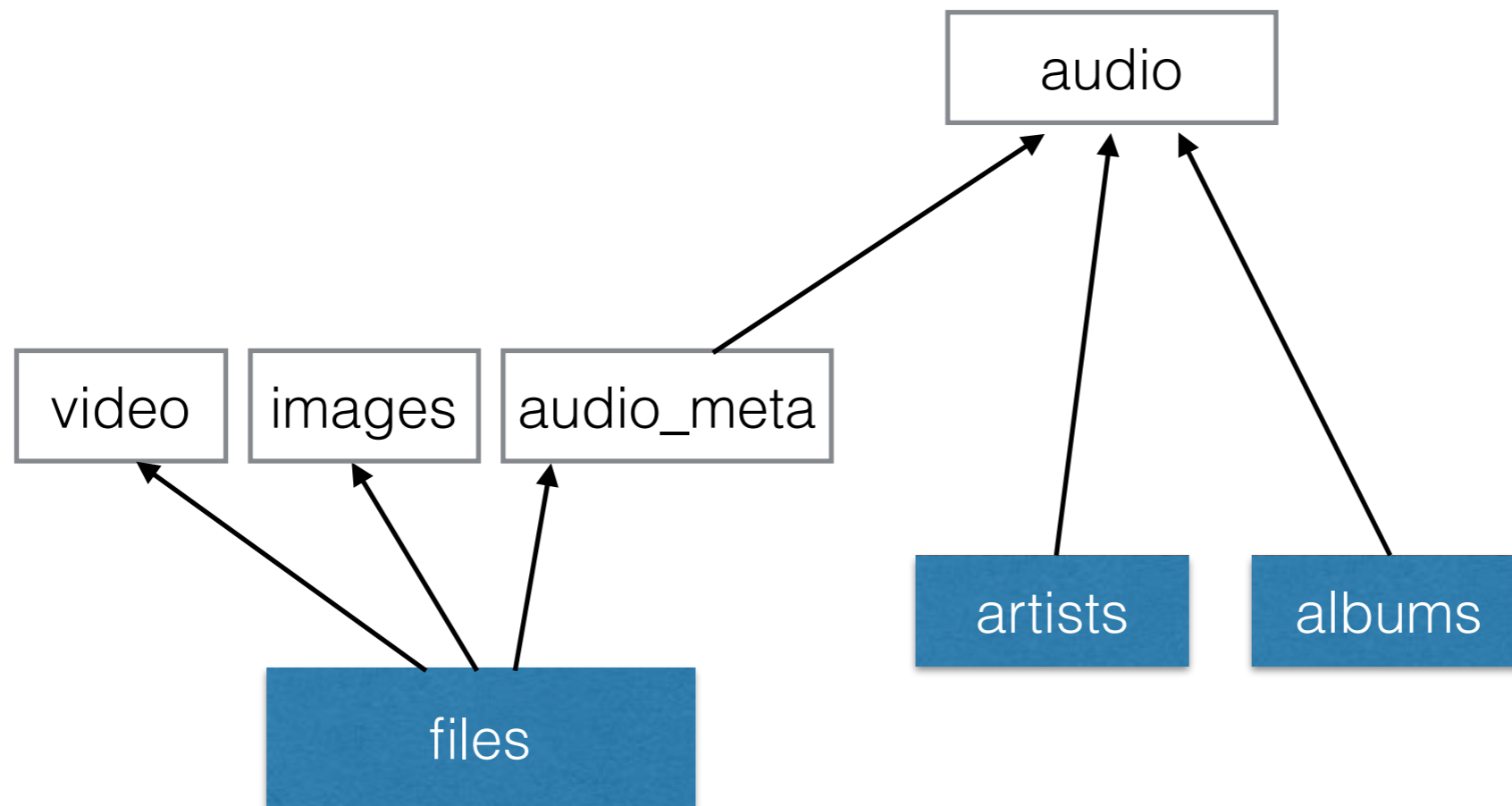

select only images

_id	_data	_size	title	...
1	a.jpg	10000	a	
2	b.bmp	20000	b	
5	image0.jpg	102400	profile	

Views in Media Provider

<view>

<table>



Views in Media Provider

```
CREATE VIEW audio_meta AS  
  SELECT _id, <audio-related columns>,  
  FROM files  
  WHERE media_type = <MEDIA_TYPE_AUDIO>;
```

```
CREATE VIEW IF NOT EXISTS audio AS  
  SELECT * FROM audio_meta  
  LEFT OUTER JOIN artists ON  
    audio_meta.artist_id=artists.artist_id  
  LEFT OUTER JOIN albums ON  
    audio_meta.album_id=albums.album_id;
```


Android SQLiteDatabase

A class to use SQLite.

```
SQLiteDatabase db = openOrCreateDatabase( "name",  
    MODE_PRIVATE, null);  
  
db.execSQL("SQL query");
```

Android SQLiteDatabase

It helps you to generate SQL statements.
query (SELECT), delete, insert, update

`db.beginTransaction(), db.endTransaction()`

`db.delete("table", "whereClause", args)`

`db.deleteDatabase(file)`

`db.insert("table", null, values)`

`db.query(...)`

`db.rawQuery("SQLquery", args)`

`db.replace("table", null, values)`

`db.update("table", values, "whereClause", args)`

Avoid using user-provided input as part of a raw query

SQL injection:

- statement =
"SELECT * FROM users WHERE name =\" +
userName + \"\";"
- If the user provides **userName** = " ' OR '1'='1 "
Statement becomes:
 - SELECT * FROM users
WHERE name = " OR '1'='1';
— always true.

Avoid using user-provided input as part of a raw query

Use ContentValues and arguments for user-provided input.

ContentValues

```
ContentValues cvalues = new ContentValues();  
cvalues.put("columnName1", value1);  
cvalues.put("columnName2", value2);  
...  
db.insert("tableName", null, cvalues);
```

- ContentValues can be optionally used as a level of abstraction for statements like INSERT, UPDATE, REPLACE

Compare to raw statements...

– Contrast with:

```
db.execSQL("INSERT INTO tableName ("  
+ columnName1 + ", " + columnName2  
+ ") VALUES (" + value1 + ", " + value2 + ")");
```

ContentValues allows you to use cleaner Java syntax rather than raw SQL syntax for some common operations.

Arguments

query(String table, String[] columns,
String **selection**, String[] **selectionArgs**,
String groupBy, String having, String orderBy)

- selection: a where clause that can contain “?”
 - type=? and date=?
- selectionArgs:
 - [“image”, “10/1/2016”]

Cursor: result of a query

Cursor lets you iterate through row results one at a time

```
— — —  
Cursor cursor = db.rawQuery("SELECT * FROM students");  
cursor.moveToFirst();  
do {  
    int id = cursor.getInt(cursor.getColumnIndex("id"));  
    String email = cursor.getString(  
        cursor.getColumnIndex("email"));  
    ...  
} while (cursor.moveToNext());  
cursor.close();
```