latex input: mmd-article-header Title: SQL Notes Author: Ethan C. Petuchowski Base Header Level: 1 latex mode: memoir Keywords: SQL, Programming Language, Syntax, Databases CSS: http://fletcherpenney.net/css/document.css xhtml header: copyright: 2014 Ethan C. Petuchowski latex input: mmd-natbib-plain latex input: mmd-article-begin-doc latex footer: mmd-memoir-footer

SQLite vs MySQL vs PostgreSQL

- All are relational database management systems (RDBMSs)
- All are open-source
- All allow you to create databases and stick tables into them
- All provide partial compliance with the ANSI SQL standard
    - But only Postgres aims for full ANSI/ISO SQL *compliance*
- Each provides a different set of datatypes
- SQLite is the simplest, Postgres is the most advanced
- Use the simplest database that will work for your project
    - Complex features are hard to configure properly
- They are all blazing fast, but not necessarily "big data" compatible
    - They use the B+-tree-based indexing system
- They each have associated GUIs to make testing and administration easier

SQLite

- Provides only a primitive set of datatypes
- Most likely, this database is good enough for your needs, and you should be glad this is the case
- SQLite is actually a C library that you [would theoretically] link into your code.
    - If using SQLiteJDBC, you will have a compiled native SQLite library for your operating system that Java interacts with using JNI, src
- It creates a single file on the OS per database
- It uses the OS to synchronize access to the database across separate processes
    - A writer will grab a write-lock on the file so that no one else can concurrently modify it
    - Readers grab a read-lock which does not lock out other readers
    - If your database file exists on a networked file system, you may have issues
- I don't think it supports user authentication or access controls
- **Don't use it if** you need a lot of concurrent writes, because SQLite doesn't support that but the other RDBMSs do

MySQL and Postgres

- These use a client-server model

- The server modifies the underlying databases
- The clients send SQL to the server, who responds based on the databases' state
- Users/access control is well-supported, and security is configurable
- They support *replication*

Postgres

- SQL standard compliant
- Complete support for ACID transactions
- Multiversion concurrency control (MVCC) instead of read locks
- Extensible using "stored procedures"
- Supports nesting of datatypes (??)
- "More reliable than the other databases" (not sure what that means)

References

- DigitalOcean

Hive

Sampling

Table Sample

- Query *samples* of the data instead of the whole table
- Can be added to any table in the from clause
- Bucket numbers start at 1
-

UNION

- *Combine results from multiple SELECT statements into a single result set*
- The number and names of columns returned by each SELECT statement must be the same

UNION ALL

Bag union; don't deduplicate rows

UNION DISTINCT

Same as just plain `UNION`

Example

```
SELECT u.id, actions.date
FROM (
        SELECT av.uid AS uid
        FROM action_video av
        WHERE av.date = '2008-06-03'
        UNION ALL
        SELECT ac.uid AS uid
        FROM action_comment ac
        WHERE ac.date = '2008-06-03'
) actions JOIN users u ON (u.id = actions.uid)
```

Make Up Your Own Table ("Scalar Query")

```
SELECT 'q' as docid, 'treasury' as term, 1 as count
```

Creates the database row

docid	term	count
q	treasury	1

as a standalone row.

RDBMS SQL

Scan vs Seek

- In the RDBMS world, scans and seeks are different ways the system reads data.
- For a **scan**, the system iterates over the *entire* table, evaluating the given predicate
- For a **seek**, we use the index to navigate directly to those rows that satisfy the predicate
- We can see what the system is going to do for a given query e.g. on Microsoft SQL Server using `showplan`

Reference

- msdn

Left Outer Join

What is the syntax?

```
SELECT *
FROM source_table LEFT OUTER JOIN target_table -- "join clause"
ON(expression) or USING(column_name) // you cannot use BOTH -- "joi
WHERE(expression) ORDER BY column names;
```

What does it do?

- ALL records in `SOURCE_TABLE` will be there
- ONLY records in the `TARGET_TABLE` matching "join condition" are kept.
- If a record for the `SOURCE_TABLE` doesn't match any records in the `TARGET_TABLE` being selected in the query, but some columns must be taken from the `TARGET_TABLE`, the value of those columns will be `NULL` in the result set.
- The word "`OUTER`" is *optional* (may be deleted with no effect)
- `USING(col)` is the *same* as saying `ON(a.col = b.col)`

Group By

From StackOverflow.

`Group By X` means put all those with the same value for X in the one group.

`Group By X, Y` means put all those with the same values for both X and Y in the one group.

Unique vs Distinct

From StackOverflow

- **`Unique`** is a keyword used in the **`Create Table()`** directive
  - to denote that a field will contain unique data
  - usually used for natural keys, foreign keys etc.
- **`Distnct`** is used in the **`Select`** statement to notify the query that you only want the unique items returned when a field holds data that may not be unique.

Foreign Keys

Sample (PostgreSQL) Code for creating a Foreign Key

```
videos_id varchar CONSTRAINT FK_videos_id
references videos(id)
match simple
on update cascade
on delete cascade
```

or simply

```
ALTER TABLE bar add FOREIGN KEY (id) references foo(id);
```

According to the Docs (which look very well-done in general):

They are by default

- Allowed to be NULL
- `MATCH SIMPLE` --- allows some foreign key columns (of a multiple column FK) to be null while other parts of the foreign key are not null.
- `on (UPDATE|DELETE) no action` --- if any referencing rows still exist when the constraint is checked, an error is raised

Though you can also do

- `MATCH FULL` --- doesn't allow one column of a multicolumn foreign key to be null unless all foreign key columns are null.
- `on (UPDATE|DELETE)` --- action possibilities are the same for both
    - `restrict` --- prevent deletion of a referenced row
        - The essential difference between these `no action` and `restrict` is that `NO ACTION` allows the check to be deferred until later in the transaction, whereas `RESTRICT` does not.
    - `cascade` --- when a referenced row is deleted, row(s) referencing it should be automatically deleted as well.
    - `set (NULL|DEFAULT)` --- update value to this value when referenced row is deleted
        - Note that these do not excuse you from observing any `constraints`. For example, if an action specifies `SET DEFAULT` but the `default` value would not satisfy the `foreign key`, the operation will fail.

NULL

- By default, values are allowed to be `NULL`
- You can prevent this by saying

    ```
    price numeric NOT NULL CHECK (price > 0)
    ```

- You may also want to specify that it *can* be `NULL` by saying

    ```
    price numeric NULL
    ```

> *Tip:* In most database designs *the majority of columns should be marked* `NOT NULL`.