Tools and Techniques

Charles Severance

www.pg4e.com/lectures/03-Techniques.sql

After CREATE TABLE

```
CREATE TABLE account (
  id SERIAL,
  email VARCHAR(128) UNIQUE,
  created_at DATE NOT NULL DEFAULT NOW(),
  updated_at DATE NOT NULL DEFAULT NOW(),
  PRIMARY KEY(id)
);

CREATE TABLE post (
  id SERIAL,
  title VARCHAR(128) UNIQUE NOT NULL,
  content VARCHAR(1024), -- Will extend with ALTER
  account_id INTEGER REFERENCES account(id) ON DELETE CASCADE,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  PRIMARY KEY(id)
);
```

```
-- Allow multiple comments
CREATE TABLE comment (
  id SERIAL,
  content TEXT NOT NULL,
  account_id INTEGER REFERENCES account(id) ON DELETE CASCADE,
  post_id INTEGER REFERENCES post(id) ON DELETE CASCADE,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  PRIMARY KEY(id)
);

CREATE TABLE fav (
  id SERIAL,
  oops TEXT,  -- Will remove later with ALTER
  post_id INTEGER REFERENCES post(id) ON DELETE CASCADE,
  account_id INTEGER REFERENCES account(id) ON DELETE CASCADE,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  UNIQUE(post_id, account_id),
  PRIMARY KEY(id)
);
```

We can adjust our schema

- Sometimes you make a mistake or your application evolves

```
CREATE TABLE fav (
  id SERIAL,
  oops TEXT,
  post_id INTEGER REFERENCES post(id) ON DELETE CASCADE,
  account_id INTEGER REFERENCES account(id) ON DELETE CASCADE,
  UNIQUE(post_id, account_id),
  PRIMARY KEY(id)
);

ALTER TABLE fav DROP COLUMN oops;
```

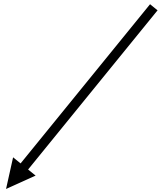
Add, Drop , Alter columns

- Can also alter indexes, uniqueness constraints, foreign keys
- Can run on a live database

```
ALTER TABLE fav DROP COLUMN oops;

ALTER TABLE post ALTER COLUMN content TYPE TEXT;

ALTER TABLE fav ADD COLUMN howmuch INTEGER;
```

Reading commands from a file

```
-- https://www.pg4e.com/lectures/03-Techniques-Load.sql

-- Start fresh - Cascade deletes it all

DELETE FROM account;
ALTER SEQUENCE account_id_seq RESTART WITH 1;
ALTER SEQUENCE post_id_seq RESTART WITH 1;
ALTER SEQUENCE comment_id_seq RESTART WITH 1;
ALTER SEQUENCE fav_id_seq RESTART WITH 1;

...
```



```
discuss=> \i 03-Techniques-load.sql
DELETE 4
ALTER SEQUENCE
ALTER SEQUENCE
ALTER SEQUENCE
ALTER SEQUENCE
INSERT 0 3
INSERT 0 3
INSERT 0 5
discuss=>
```

Dates

Date Types (Review)

- DATE - 'YYYY-MM-DD'
- TIME - 'HH:MM:SS'
- TIMESTAMP - 'YYYY-MM-DD HH:MM:SS' (4713 BC, 294276 AD)
- TIMESTAMPTZ – "TIMESTAMP WITH TIME ZONE"
- Built-in PostgreSQL function NOW()

Setting default values

- We can save some code by auto-populating date fields when a row is INSERTed
- We will auto-set on UPDATEs later…

```
CREATE TABLE fav (
  id SERIAL,
  post_id INTEGER REFERENCES post(id) ON DELETE CASCADE,
  account_id INTEGER REFERENCES account(id) ON DELETE CASCADE,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  UNIQUE(post_id, account_id),
  PRIMARY KEY(id)
);
```

TIMESTAMPTZ – Best Practice

- Store time stamps with timezone
- Prefer UTC for stored time stamps
- Convert to local time zone when retrieving

```
discuss=> SELECT NOW(), NOW() AT TIME ZONE 'UTC', NOW() AT TIME ZONE 'HST';
              now               |          timezone          |          timezone
--------------------------------+----------------------------+----------------------------
 2019-06-10 11:42:51.52127-04 | 2019-06-10 15:42:51.52127 | 2019-06-10 05:42:51.52127
```

PostgreSQL time zones

```
discuss=> SELECT * FROM pg_timezone_names;

               name                 | abbrev | utc_offset | is_dst
------------------------------------+--------+------------+--------
 Indian/Mauritius                   | +04    | 04:00:00   | f
 Indian/Chagos                      | +06    | 06:00:00   | f
 Indian/Mayotte                     | EAT    | 03:00:00   | f
 Indian/Christmas                   | +07    | 07:00:00   | f
 Indian/Cocos                       | +0630  | 06:30:00   | f
 Indian/Comoro                      | EAT    | 03:00:00   | f

discuss=> SELECT * FROM pg_timezone_names WHERE name LIKE '%Hawaii%';
   name     | abbrev | utc_offset | is_dst
 -----------+--------+------------+--------
 US/Hawaii | HST    | -10:00:00  | f
```

Casting to different types

- We use the phrase 'casting' to mean convert from one type to another
- Postgres has several forms of casting

```
postgres=# SELECT NOW()::DATE, CAST(NOW() AS DATE), CAST(NOW() AS TIME);
    now     |    now     |       now
------------+------------+-----------------
 2019-06-10 | 2019-06-10 | 11:45:42.899995
```

Intervals

- We can do date interval arithmetic

```
postgres=# SELECT NOW(), NOW() - INTERVAL '2 days', (NOW() - INTERVAL '2 days')::DATE;
              now               |           ?column?            |    date
--------------------------------+-------------------------------+------------
 2019-06-10 11:56:34.886679-04 | 2019-06-08 11:56:34.886679-04 | 2019-06-08
```

Using date_trunc()

- Sometimes we want to discard some of the accuracy that is in a TIMESTAMP

```
discuss=> SELECT id, content, created_at FROM comment
discuss->     WHERE created_at >= DATE_TRUNC('day',NOW())
discuss->     AND created_at < DATE_TRUNC('day',NOW() + INTERVAL '1 day');

 id |                          content                          | created_at
----+-----------------------------------------------------------+------------
 11 | I agree                                                   | 2019-06-10
 12 | Especially for counting                                   | 2019-06-10
 13 | And I don't understand why                                | 2019-06-10
 14 | Someone should make "EasySoup" or something like that | 2019-06-10
 15 | Good idea - I might just do that                          | 2019-06-10
```

Performance: Table Scans

- Not all equivalent queries have the same performance

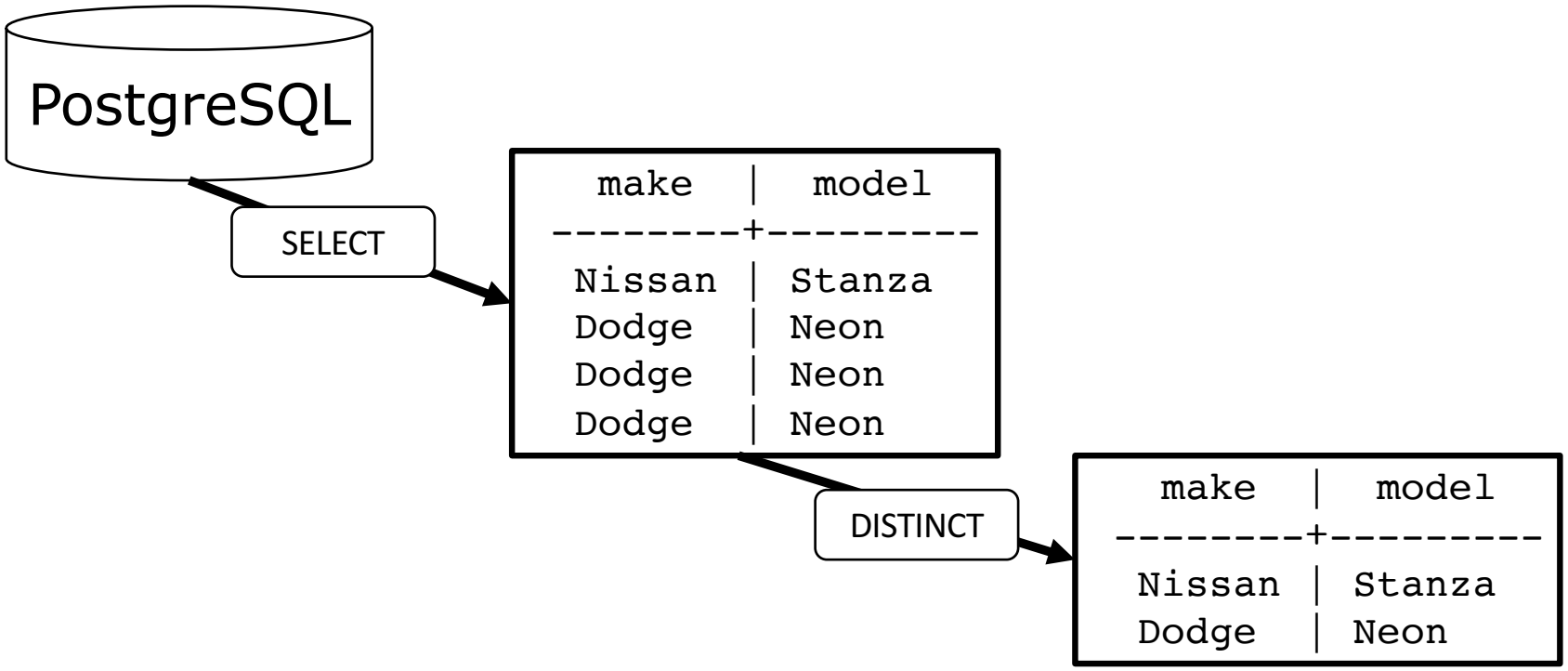
```
discuss=> SELECT id, content, created_at FROM comment
discuss->   WHERE created_at::DATE = NOW()::DATE;

 id |                          content                          | created_at
----+-----------------------------------------------------------+------------
 11 | I agree                                                   | 2019-06-10
 12 | Especially for counting                                   | 2019-06-10
 13 | And I don't understand why                                | 2019-06-10
 14 | Someone should make "EasySoup" or something like that | 2019-06-10
 15 | Good idea - I might just do that                          | 2019-06-10
```

DISTINCT / GROUP BY

Reducing the result set

- DISTINCT only returns unique rows in a result set – and row will only appear once
- DISTINCT ON limits duplicate removal to a set of columns
- GROUP BY is combined with aggregate functions like COUNT(), MAX(), SUM(), AVE() …

Reducing a result set



Racing Data


```
discuss=> select * from racing;
```

make	model	year	price
Nissan	Stanza	1990	2000
Dodge	Neon	1995	800
Dodge	Neon	1998	2500
Dodge	Neon	1999	3000
Ford	Mustang	2001	1000
Ford	Mustang	2005	2000
Subaru	Impreza	1997	1000
Mazda	Miata	2001	5000
Mazda	Miata	2001	3000
Mazda	Miata	2001	2500
Mazda	Miata	2002	5500
Opel	GT	1972	1500
Opel	GT	1969	7500
Opel	Cadet	1973	500

make	model	year	price
Nissan	Stanza	1990	2000
Dodge	Neon	1995	800
Dodge	Neon	1998	2500
Dodge	Neon	1999	3000
Ford	Mustang	2001	1000
Ford	Mustang	2005	2000
Subaru	Impreza	1997	1000
Mazda	Miata	2001	5000
Mazda	Miata	2001	3000
Mazda	Miata	2001	2500
Mazda	Miata	2002	5500
Opel	GT	1972	1500
Opel	GT	1969	7500
Opel	Cadet	1973	500

```
SELECT DISTINCT model FROM racing;
```

model
Stanza
Neon
Mustang
Impreza
Miata
GT
Cadet

make	model	year	price
Nissan	Stanza	1990	2000
Dodge	Neon	1995	800
Dodge	Neon	1998	2500
Dodge	Neon	1999	3000
Ford	Mustang	2001	1000
Ford	Mustang	2005	2000
Subaru	Impreza	1997	1000
Mazda	Miata	2001	5000
Mazda	Miata	2001	3000
Mazda	Miata	2001	2500
Mazda	Miata	2002	5500
Opel	GT	1972	1500
Opel	GT	1969	7500
Opel	Cadet	1973	500

```
SELECT DISTINCT ON (model)
  make,model FROM racing;
```

make	model
Opel	Cadet
Opel	GT
Subaru	Impreza
Mazda	Miata
Ford	Mustang
Dodge	Neon
Nissan	Stanza

Lets play with time zones

```
discuss=> SELECT * FROM pg_timezone_names;

               name                 | abbrev | utc_offset | is_dst
------------------------------------+--------+------------+--------
 Indian/Mauritius                   | +04    | 04:00:00   | f
 Indian/Chagos                      | +06    | 06:00:00   | f
 Indian/Mayotte                     | EAT    | 03:00:00   | f
 Indian/Christmas                   | +07    | 07:00:00   | f
 Indian/Cocos                       | +0630  | 06:30:00   | f
 Indian/Comoro                      | EAT    | 03:00:00   | f

discuss=> SELECT * FROM pg_timezone_names WHERE name LIKE '%Hawaii%';
   name     | abbrev | utc_offset | is_dst
 -----------+--------+------------+--------
 US/Hawaii | HST    | -10:00:00  | f
```

Aggregate / GROUP BY

```
discuss=> SELECT COUNT(abbrev), abbrev FROM pg_timezone_names GROUP BY abbrev;
 count | abbrev
-------+--------
     2 | +00
     1 | PST
     4 | IST
     2 | -01
     4 | HST
     6 | +09
    15 | +05
     7 | ADT
     1 | -12
...
```

HAVING clause

```
discuss=> SELECT COUNT(abbrev) AS ct, abbrev FROM  pg_timezone_names
discuss-> WHERE is_dst= 't' GROUP BY abbrev HAVING COUNT(abbrev) > 10;

 ct | abbrev
----+--------
 12 | PDT
 22 | EEST
 24 | CDT
 36 | CEST
 28 | EDT
 15 | MDT
```

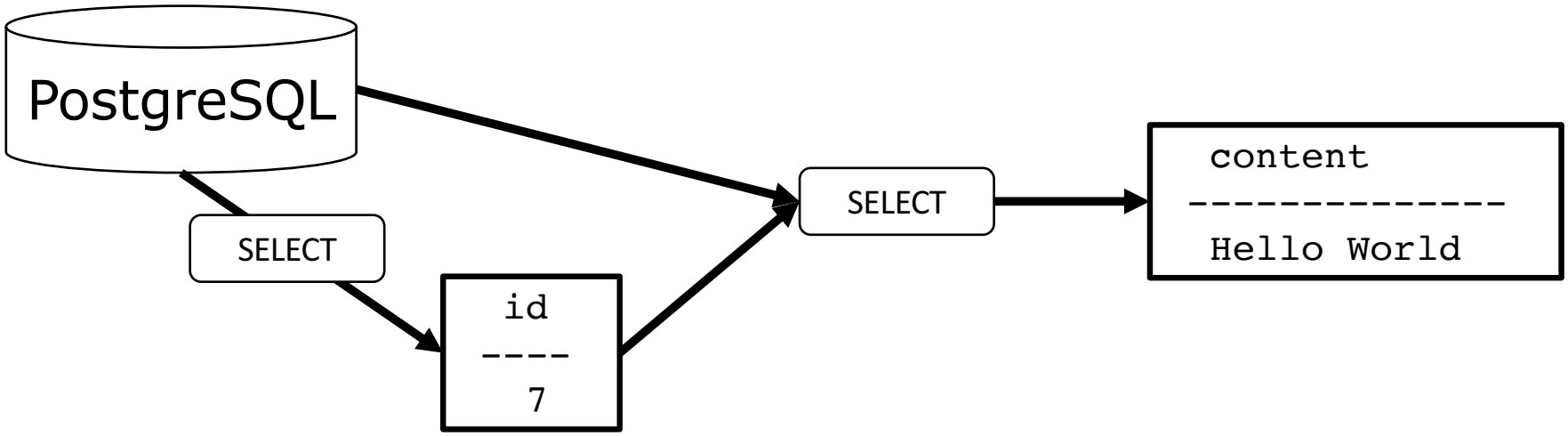
Sub-Queries

A query within an query

- Can use a value or set of values in a query that are computed by another query

```
SELECT * FROM account
 WHERE email='ed@umich.edu';

SELECT content FROM comment
 WHERE account_id = 7;

SELECT content FROM comment
 WHERE account_id = (SELECT id FROM account WHERE email='ed@umich.edu');
```

Sub Query



HAVING clause

```
discuss=> SELECT COUNT(abbrev) AS ct, abbrev FROM  pg_timezone_names
discuss->   WHERE is_dst= 't' GROUP BY abbrev HAVING COUNT(abbrev) > 10;

 ct | abbrev
----+--------
 12 | PDT
 22 | EEST
 24 | CDT
 36 | CEST
 28 | EDT
 15 | MDT
```

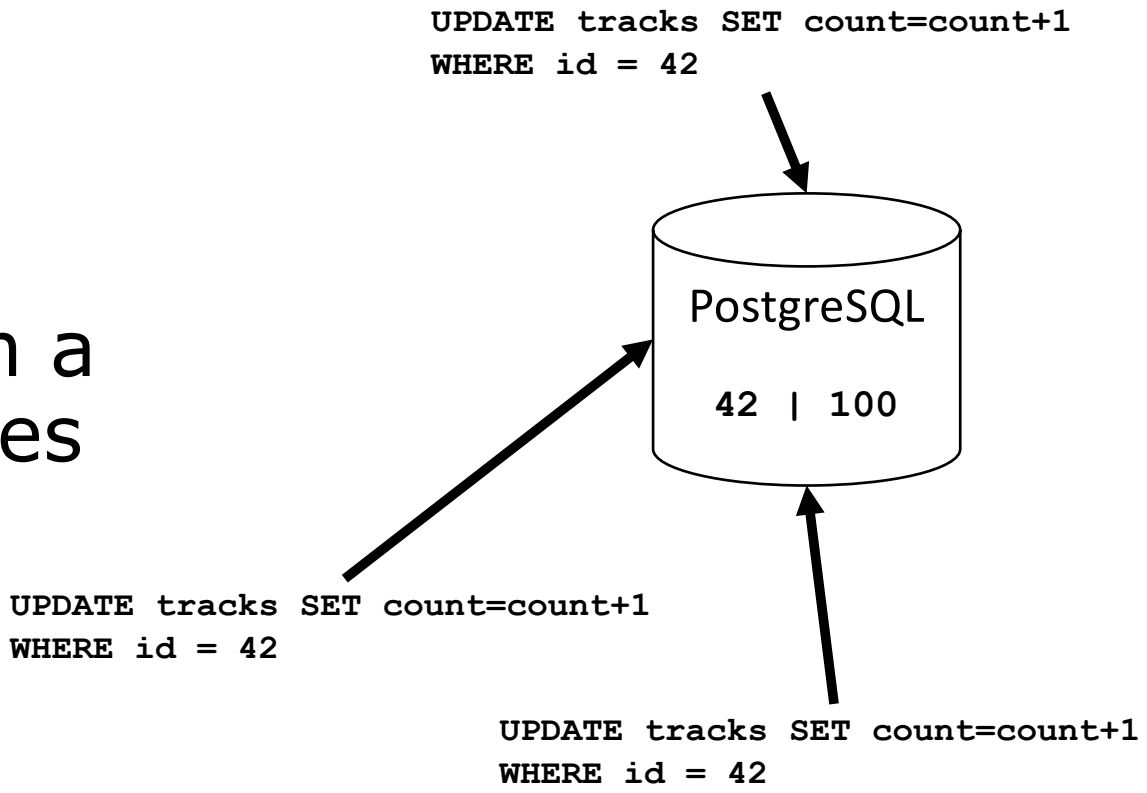
Using a Sub-Query

```
discuss=> SELECT ct, abbrev FROM
discuss-> (
discuss->   SELECT COUNT(abbrev) AS ct, abbrev
discuss->   FROM  pg_timezone_names
discuss->   WHERE is_dst = 't' GROUP BY abbrev
discuss-> ) AS zap
discuss-> WHERE ct > 10;
 ct | abbrev
----+--------
 12 | PDT
 22 | EEST
 24 | CDT
 36 | CEST
 28 | EDT
 15 | MDT
```

Concurrency

Concurrency

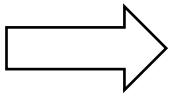
- Databases are designed to accept SQL commands from a variety of sources simultaneously and perform them atomically



Transactions and Atomicity

- To implement atomicity, PostgreSQL "locks" areas before it starts an SQL command that might change an area of the database
- All other access to that area must wait until the area is unlocked

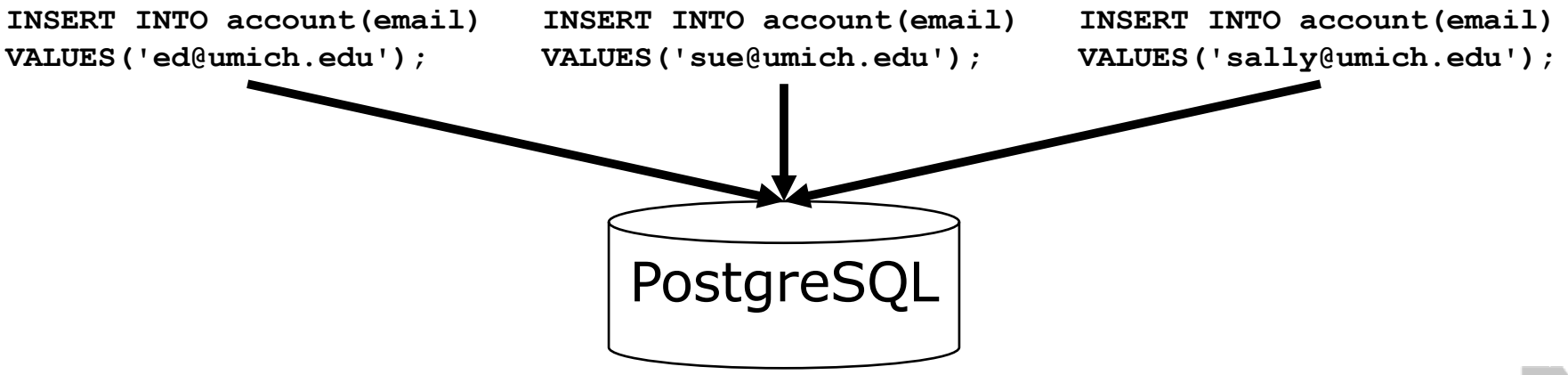
```
UPDATE tracks
SET count=count+1
WHERE id = 42
```



```
LOCK ROW 42 OF tracks
READ count FROM tracks ROW 42
count = count + 1
WRITE count TO tracks ROW 42
UNLOCK ROW 42 OF tracks
```

Single SQL Statements are Atomic

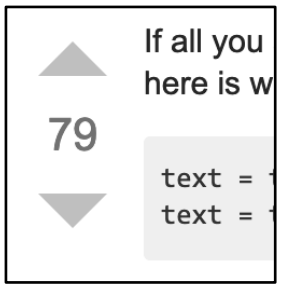
- All the inserts will work and get a unique primary key
- Which account gets which key is not predictable



Compound Statements

- There are statements which do more than one thing in one statement for efficiency and concurrency.

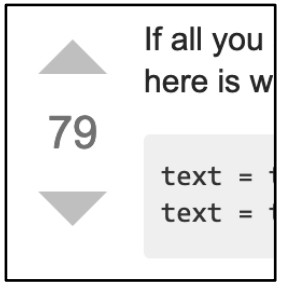
```
INSERT INTO fav (post_id, account_id, howmuch)
  VALUES (1,1,1)
RETURNING *;

UPDATE fav SET howmuch=howmuch+1
  WHERE post_id = 1 AND account_id = 1
RETURNING *;
```



ON CONFLICT

- Sometimes you "bump into" a constraint on purpose

```
-- This will fail
INSERT INTO fav (post_id, account_id, howmuch)
  VALUES (1,1,1)
RETURNING *;

INSERT INTO fav (post_id, account_id, howmuch)
  VALUES (1,1,1)
  ON CONFLICT (post_id, account_id)
  DO UPDATE SET howmuch = fav.howmuch + 1
RETURNING *;
```



Multi-Statement Transactions

```
BEGIN;
SELECT howmuch FROM fav WHERE account_id=1 AND post_id=1 FOR UPDATE OF fav;
-- Time passes...
UPDATE SET howmuch=999 WHERE account_id=1 AND post_id=1;
ROLLBACK;
SELECT howmuch FROM fav WHERE account_id=1 AND post_id=1;
```

```
BEGIN;
SELECT howmuch FROM fav WHERE account_id=1 AND post_id=1 FOR UPDATE OF fav;
-- Time passes...
UPDATE SET howmuch=999 WHERE account_id=1 AND post_id=1;
COMMIT;
SELECT howmuch FROM fav WHERE account_id=1 AND post_id=1;
```

Play with this with two windows open ☺

Transactions and Performance

- The implementation of transactions makes a big difference in database performance
  - Lock granularity
  - Lock implementation

Transaction Topics

- Lock strength UPDATE, NO KEY UPDATE
- What to do when encountering a lock (WAIT), NOWAIT, SKIP LOCKED

Stored Procedures

Stored Procedures

- A stored procedure is a bit of reusable code that runs inside of the database server
- Technically there are multiple language choices but just use "plpgsql"
- Generally quite non-portable
- Usually the goal is to have fewer SQL statements

Stored Procedures

- You should have a strong reason to use a stored procedure
  - Major performance problem
  - Harder to test / modify
  - No database portability
  - Some rule that *must* be enforced

Recall

```
CREATE TABLE fav (
  id SERIAL,
  post_id INTEGER REFERENCES post(id) ON DELETE CASCADE,
  account_id INTEGER REFERENCES account(id) ON DELETE CASCADE,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  UNIQUE(post_id, account_id),
  PRIMARY KEY(id)
);

UPDATE fav SET howmuch=howmuch+1
  WHERE post_id = 1 AND account_id = 1;


UPDATE fav SET howmuch=howmuch+1, updated_at=NOW()
  WHERE post_id = 1 AND account_id = 1;
```

Using a trigger for updated_at

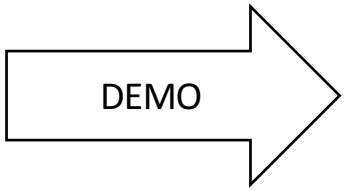
```
CREATE OR REPLACE FUNCTION trigger_set_timestamp()
RETURNS TRIGGER AS $$
BEGIN
  NEW.updated_at = NOW();
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER set_timestamp
BEFORE UPDATE ON fav
FOR EACH ROW
EXECUTE PROCEDURE trigger_set_timestamp();


UPDATE fav SET howmuch=howmuch+1
  WHERE post_id = 1 AND account_id = 1;
```

DEMO
Reading and Parsing Files

CSV -> Normalized Database

03-Techniques.csv :

```
Zap,A
Zip,A
One,B
Two,B
```




```
discuss=> select * from xy;
 id |  x  | y_id
----+-----+------
  1 | Zap |    2
  2 | Zip |    2
  3 | One |    1
  4 | Two |    1
```

```
discuss=> select * from y;
 id | y
----+---
  1 | B
  2 | A
```

Summary

https://www.postgresql.org/docs/11/sql-select.html

```
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
[ * | expression [ [ AS ] output_name ] [, ...] ]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY grouping_element [, ...] ]
[ HAVING condition [, ...]
[ ORDER BY expression [ ASC | DESC | USING operator ]
[ LIMIT { count | ALL } ] [ OFFSET start [ ROW | ROWS ] ]
[ FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE }
    [ OF table_name [, ...] ] [ NOWAIT | SKIP LOCKED ] [...] ]
```

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

Insert new Contributors and Translators here including names and dates

Continue new Contributors and Translators here