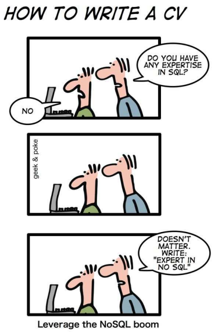
BT5110 Week 12 Lecture
SQL, NoSQL & NewSQL

Chan Chee Yong
chancy@comp.nus.edu.sg


HOW TO WRITE A CV
DO YOU HAVE ANY EXPERTISE IN SQL?
NO
geek & poke
DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"
Leverage the NoSQL boom

Modern DBMSs: NoSQL & NewSQL

AEROSPIKE
cassandra
Clustrix
Couchbase
amazon DynamoDB
APACHE HBASE
IBM Cloudant
MarkLogic
memsql Speed. Scale. Simplicity.
mongoDB
NUODB
ORACLE NOSQL DATABASE
riak
splice MACHINE
TRANSLATTICE
VOLTDB

http://www.informationweek.com/big-data/big-data-analytics/16-nosql-newsql-databases-to-watch/d/d-id/1269559

Relational Database Systems

(Image: Software Engineering Daily)

Transactions

- Abstraction for representing a logical unit of work
- **ACID Properties**
  - Atomicity: Either all the effects of the transactions are reflected in the database or none are
  - Consistency: The execution of a transaction in isolation preserves the consistency of the database
  - Isolation: The execution of a transaction is isolated from the effects of other concurrent transaction executions
  - Durability: The effects of a committed transaction persists in the database even in the presence of system failures

Transaction Example: Money Transfer

```
int Transfer (int fromAcctId, int toAcctId, int amount)
{
    EXEC SQL BEGIN DECLARE SECTION;
        int fromBalance;    int toBalance;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL WHENEVER SQLERROR GOTO query_error;

    EXEC SQL SELECT balance INTO :fromBalance FROM Accounts
        WHERE accountId = :fromAcctId;
    if (fromBalance < amount) {
        EXEC SQL ROLLBACK;   return 1;
    }
    EXEC SQL SELECT balance INTO :toBalance FROM Accounts
        WHERE accountId = :toAcctId;
    EXEC SQL UPDATE Accounts SET balance = :toBalance + :amount
        WHERE accountId = :toAcctId;
    EXEC SQL UPDATE Accounts SET balance = :fromBalance - :amount
        WHERE accountId = :fromAcctId;
    EXEC SQL COMMIT;
    return 0;
    query_error: printf ("SQL error: %ld\n", sqlca->sqlcode); exit();
}
```

Transaction Example: Money Transfer

Two possible execution outcomes:

Abort

```
begin transaction;

select      balance into :fromBalance
from        Accounts
where       accountId = :fromAcctId;

rollback;
```

Commit

```
begin transaction;

select      balance into :fromBalance
from        Accounts
where       accountId = :fromAcctId;

select      balance into :toBalance
from        Accounts
where       accountId = :toAcctId;

update      Accounts
set         balance = :toBalance + :amount
where       accountId = 1;

update      Accounts
set         balance = :fromBalance - :amount
where       accountId = 2;

commit;
```

Serial Transaction Executions

Two possible serial executions of Transfer(1,2,100) & Transfer(2,1,100)

(1):
```
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;
update Accounts set balance = :frombal - 100 where accountId = 1;
commit;
begin transaction;
select balance into :frombal from Accounts where accountId = 2;
select balance into :tobal from Accounts where accountId = 1;
update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
```

(2):
```
begin transaction;
select balance into :frombal from Accounts where accountId = 2;
select balance into :tobal from Accounts where accountId = 1;
update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;
update Accounts set balance = :frombal - 100 where accountId = 1;
commit;
```

Concurrent Transaction Executions

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;
begin transaction;
select balance into :frombal from Accounts where accountId = 2;
select balance into :tobal from Accounts where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 1;
commit;
update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
```

Serializable Transaction Executions

- A concurrent execution of a set of transactions is serializable if this execution is equivalent to some serial execution
- A concurrent execution *CE* is **equivalent** to a serial execution *SE* if
  - both *CE* & *SE* produce the same final database state, and
  - every read operation in *CE* returns the same value as the corresponding read operation in *SE*
- A serial execution is trivially serializable

Example 1: Non-serializable Execution

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;

begin transaction;
select balance into :frombal from Accounts where accountId = 2;
select balance into :tobal from Accounts where accountId = 1;

update Accounts set balance = :frombal - 100 where accountId = 1;
commit;

update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
```

Transfer(1,2,100)

frombal:
tobal:

Accounts

(1, $400)
(2, $2000)

Transfer(2,1,100)

frombal:
tobal:

Example 1: Non-serializable Execution

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;

begin transaction;
select balance into :frombal from Accounts where accountId = 2;
select balance into :tobal from Accounts where accountId = 1;

update Accounts set balance = :frombal - 100 where accountId = 1;
commit;

update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
```

Transfer(1,2,100)

frombal: 400
tobal:

Accounts

(1, $400)
(2, $2000)

Transfer(2,1,100)

frombal:
tobal:

Example 1: Non-serializable Execution

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;

begin transaction;
select balance into :frombal from Accounts where accountId = 2;
select balance into :tobal from Accounts where accountId = 1;

update Accounts set balance = :frombal - 100 where accountId = 1;
commit;

update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
```

Transfer(1,2,100)

frombal: 400
tobal: 2000

Accounts

(1, \$400)
(2, \$2000)

Transfer(2,1,100)

frombal:
tobal:

Example 1: Non-serializable Execution

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;

begin transaction;
select balance into :frombal from Accounts where accountId = 2;
select balance into :tobal from Accounts where accountId = 1;

update Accounts set balance = :frombal - 100 where accountId = 1;
commit;

update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
```

Transfer(1,2,100)

frombal: 400
tobal: 2000

Accounts

(1, \$400)
(2, \$2100)

Transfer(2,1,100)

frombal:
tobal:

Example 1: Non-serializable Execution

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

begin transaction;
select balance **into** :frombal **from** Accounts **where** accountId = 1;
select balance **into** :tobal **from** Accounts **where** accountId = 2;
update Accounts **set** balance = :tobal + 100 **where** accountId = 2;

begin transaction;
select balance **into** :frombal **from** Accounts **where** accountId = 2;
select balance **into** :tobal **from** Accounts **where** accountId = 1;

update Accounts **set** balance = :frombal - 100 **where** accountId = 1;
commit;

update Accounts **set** balance = :tobal + 100 **where** accountId = 1;
update Accounts **set** balance = :frombal - 100 **where** accountId = 2;
commit;

Transfer(1,2,100)

frombal: 400
tobal: 2000

Accounts

(1, $400)
(2, $2100)

Transfer(2,1,100)

frombal: 2100
tobal:

Example 1: Non-serializable Execution

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

begin transaction;
select balance **into** :frombal **from** Accounts **where** accountId = 1;
select balance **into** :tobal **from** Accounts **where** accountId = 2;
update Accounts **set** balance = :tobal + 100 **where** accountId = 2;

begin transaction;
select balance **into** :frombal **from** Accounts **where** accountId = 2;
select balance **into** :tobal **from** Accounts **where** accountId = 1;

update Accounts **set** balance = :frombal - 100 **where** accountId = 1;
commit;

update Accounts **set** balance = :tobal + 100 **where** accountId = 1;
update Accounts **set** balance = :frombal - 100 **where** accountId = 2;
commit;

Transfer(1,2,100)

frombal: 400
tobal: 2000

Accounts

(1, \$400)
(2, \$2100)

Transfer(2,1,100)

frombal: 2100
tobal: 400

Example 1: Non-serializable Execution

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;

begin transaction;
select balance into :frombal from Accounts where accountId = 2;
select balance into :tobal from Accounts where accountId = 1;

update Accounts set balance = :frombal - 100 where accountId = 1;
commit;

update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
```

Transfer(1,2,100)

frombal: 400
tobal: 2000

Accounts

(1, \$300)
(2, \$2100)

Transfer(2,1,100)

frombal: 2100
tobal: 400

Example 1: Non-serializable Execution

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;

begin transaction;
select balance into :frombal from Accounts where accountId = 2;
select balance into :tobal from Accounts where accountId = 1;

update Accounts set balance = :frombal - 100 where accountId = 1;
commit;

update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
```

Transfer(1,2,100)

frombal: 400
tobal: 2000

Accounts

(1, $500)
(2, $2100)

Transfer(2,1,100)

frombal: 2100
tobal: 400

Example 1: Non-serializable Execution

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;

begin transaction;
select balance into :frombal from Accounts where accountId = 2;
select balance into :tobal from Accounts where accountId = 1;

update Accounts set balance = :frombal - 100 where accountId = 1;
commit;

update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
```

Transfer(1,2,100)

frombal: 400
tobal: 2000

Accounts

(1, $500)
(2, $2000)

Transfer(2,1,100)

frombal: 2100
tobal: 400

Example 1: Non-serializable Execution

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;

begin transaction;
select balance into :frombal from Accounts where accountId = 2;
select balance into :tobal from Accounts where accountId = 1;

update Accounts set balance = :frombal - 100 where accountId = 1;
commit;

update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
```

Transfer(1,2,100)

frombal: 400
tobal: 2000

Accounts

(1, $500)
(2, $2000)

Transfer(2,1,100)

frombal: 2100
tobal: 400

Example 2: Serializable Execution

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;

begin transaction;
select balance into :frombal from Accounts where accountId = 2;

update Accounts set balance = :frombal - 100 where accountId = 1;
commit;

select balance into :tobal from Accounts where accountId = 1;
update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
```

Transfer(1,2,100)

```
frombal:
tobal:
```

Accounts

```
(1, $400)
(2, $2000)
```

Transfer(2,1,100)

```
frombal:
tobal:
```

Example 2: Serializable Execution

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;

begin transaction;
select balance into :frombal from Accounts where accountId = 2;

update Accounts set balance = :frombal - 100 where accountId = 1;
commit;

select balance into :tobal from Accounts where accountId = 1;
update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
```

Transfer(1,2,100)

frombal: 400
tobal:

Accounts

(1, \$400)
(2, \$2000)

Transfer(2,1,100)

frombal:
tobal:

Example 2: Serializable Execution

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

begin transaction;
select balance **into** :frombal **from** Accounts **where** accountId = 1;
select balance **into** :tobal **from** Accounts **where** accountId = 2;
update Accounts **set** balance = :tobal + 100 **where** accountId = 2;

begin transaction;
select balance **into** :frombal **from** Accounts **where** accountId = 2;

update Accounts **set** balance = :frombal - 100 **where** accountId = 1;
commit;

select balance **into** :tobal **from** Accounts **where** accountId = 1;
update Accounts **set** balance = :tobal + 100 **where** accountId = 1;
update Accounts **set** balance = :frombal - 100 **where** accountId = 2;
commit;

Transfer(1,2,100)

frombal: 400
tobal: 2000

Accounts

(1, $400)
(2, $2000)

Transfer(2,1,100)

frombal:
tobal:

Example 2: Serializable Execution

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;

begin transaction;
select balance into :frombal from Accounts where accountId = 2;

update Accounts set balance = :frombal - 100 where accountId = 1;
commit;

select balance into :tobal from Accounts where accountId = 1;
update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
```

Transfer(1,2,100)

frombal: 400
tobal: 2000

Accounts

(1, $400)
(2, $2100)

Transfer(2,1,100)

frombal:
tobal:

Example 2: Serializable Execution

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;

begin transaction;
select balance into :frombal from Accounts where accountId = 2;

update Accounts set balance = :frombal - 100 where accountId = 1;
commit;

select balance into :tobal from Accounts where accountId = 1;
update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
```

Transfer(1,2,100)

frombal: 400
tobal: 2000

Accounts

(1, $400)
(2, $2100)

Transfer(2,1,100)

frombal: 2100
tobal:

Example 2: Serializable Execution

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

begin transaction;
select balance **into** :frombal **from** Accounts **where** accountId = 1;
select balance **into** :tobal **from** Accounts **where** accountId = 2;
update Accounts **set** balance = :tobal + 100 **where** accountId = 2;

begin transaction;
select balance **into** :frombal **from** Accounts **where** accountId = 2;

update Accounts **set** balance = :frombal - 100 **where** accountId = 1;
commit;

select balance **into** :tobal **from** Accounts **where** accountId = 1;
update Accounts **set** balance = :tobal + 100 **where** accountId = 1;
update Accounts **set** balance = :frombal - 100 **where** accountId = 2;
commit;

Transfer(1,2,100)

frombal: 400
tobal: 2000

Accounts

(1, $300)
(2, $2100)

Transfer(2,1,100)

frombal: 2100
tobal:

Example 2: Serializable Execution

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;

begin transaction;
select balance into :frombal from Accounts where accountId = 2;

update Accounts set balance = :frombal - 100 where accountId = 1;
commit;

select balance into :tobal from Accounts where accountId = 1;
update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
```

Transfer(1,2,100)

```
frombal: 400
tobal: 2000
```

Accounts

```
(1, $300)
(2, $2100)
```

Transfer(2,1,100)

```
frombal: 2100
tobal: 300
```

Example 2: Serializable Execution

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;

begin transaction;
select balance into :frombal from Accounts where accountId = 2;

update Accounts set balance = :frombal - 100 where accountId = 1;
commit;

select balance into :tobal from Accounts where accountId = 1;
update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
```

Transfer(1,2,100)

frombal: 400
tobal: 2000

Accounts

(1, $400)
(2, $2100)

Transfer(2,1,100)

frombal: 2100
tobal: 300

Example 2: Serializable Execution

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select balance into :frombal from Accounts where accountId = 1;
select balance into :tobal from Accounts where accountId = 2;
update Accounts set balance = :tobal + 100 where accountId = 2;

begin transaction;
select balance into :frombal from Accounts where accountId = 2;

update Accounts set balance = :frombal - 100 where accountId = 1;
commit;

select balance into :tobal from Accounts where accountId = 1;
update Accounts set balance = :tobal + 100 where accountId = 1;
update Accounts set balance = :frombal - 100 where accountId = 2;
commit;
```

Transfer(1,2,100)

frombal: 400
tobal: 2000

Accounts

(1, \$400)
(2, \$2000)

Transfer(2,1,100)

frombal: 2100
tobal: 300

Example 2: Serializable Execution

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

begin transaction;
select balance **into** :frombal **from** Accounts **where** accountId = 1;
select balance **into** :tobal **from** Accounts **where** accountId = 2;
update Accounts **set** balance = :tobal + 100 **where** accountId = 2;

begin transaction;
select balance **into** :frombal **from** Accounts **where** accountId = 2;

update Accounts **set** balance = :frombal - 100 **where** accountId = 1;
commit;

select balance **into** :tobal **from** Accounts **where** accountId = 1;
update Accounts **set** balance = :tobal + 100 **where** accountId = 1;
update Accounts **set** balance = :frombal - 100 **where** accountId = 2;
commit;

Transfer(1,2,100)

frombal: 400
tobal: 2000

Accounts

(1, \$400)
(2, \$2000)

Transfer(2,1,100)

frombal: 2100
tobal: 300

SQL Isolation Levels

- The isolation level for a transaction affects what the transaction will read
- SQL defines four isolation levels
  - Read Uncommitted (weakest isolation level)
  - Read Committed
  - Repeatable Read
  - Serializable (strongest isolation level)
- Choice of isolation level affects correctness vs performance tradeoff
- In many DBMSs, the default isolation level is Read Commited
- Configure using **set transaction isolation level** statement

Creating Indexes to Speed up Queries

- Consider the following SQL query:

```
SELECT *
FROM    Customers
WHERE  state = ’CA’ AND city = ’Santa Barbara’
```

- How to create an index on attributes (state,city) of Customers table?

```
CREATE INDEX ON Customers (state,city)
```

Creating Indexes to Speed up Queries (cont.)

- Index on (state,city) could speed up the following queries:
  - SELECT * FROM Customers WHERE state = ’CA’ AND city = ’Napa’
  - SELECT * FROM Customers WHERE state $=$ ’CA’ AND city $<$ ’Napa’
  - SELECT * FROM Customers WHERE state $=$ ’CA’ AND city $>$ ’Napa’
  - SELECT * FROM Customers WHERE state $>=$ ’CA’
- However, the index is not useful for these queries:
  - SELECT * FROM Customers WHERE city = ’Springfield’
  - SELECT * FROM Customers WHERE state $>$ ’CA’ AND city $=$ ’Napa’

Scalar Subqueries

Sales

name	month	amount
Alice	2018-01-01	400
Alice	2018-02-01	300
Alice	2018-03-01	600
Bob	2018-01-01	200
Bob	2018-02-01	250
Bob	2018-03-01	300
Carol	2018-01-01	250
Carol	2018-02-01	200
Carol	2018-03-01	400

name	month	amount	total
Alice	2018-01-01	400	1300
Alice	2018-02-01	300	1300
Alice	2018-03-01	600	1300
Bob	2018-01-01	200	750
Bob	2018-02-01	250	750
Bob	2018-03-01	300	750
Carol	2018-01-01	250	850
Carol	2018-02-01	200	850
Carol	2018-03-01	400	850

```
SELECT     *, (SELECT SUM(S2.amount)
              FROM Sales S2
              WHERE S2.name = S.name)
FROM       Sales S
ORDER BY   name, month
```

Common Table Expressions (CTEs)

Sales

name	month	amount
Alice	2018-01-01	400
Alice	2018-02-01	300
Alice	2018-03-01	600
Bob	2018-01-01	200
Bob	2018-02-01	250
Bob	2018-03-01	300
Carol	2018-01-01	250
Carol	2018-02-01	200
Carol	2018-03-01	400

name	month	amount	total
Alice	2018-01-01	400	1300
Alice	2018-02-01	300	1300
Alice	2018-03-01	600	1300
Bob	2018-01-01	200	750
Bob	2018-02-01	250	750
Bob	2018-03-01	300	750
Carol	2018-01-01	250	850
Carol	2018-02-01	200	850
Carol	2018-03-01	400	850

```
WITH TotalSales AS
        (SELECT name, SUM(amount) AS total
        FROM Sales
        GROUP BY name)
SELECT      S.*, T.total
FROM        Sales S NATUAL JOIN TotalSales T
ORDER BY S.name, S.month
```

Window Functions

Sales

name	month	amount
Alice	2018-01-01	400
Alice	2018-02-01	300
Alice	2018-03-01	600
Bob	2018-01-01	200
Bob	2018-02-01	250
Bob	2018-03-01	300
Carol	2018-01-01	250
Carol	2018-02-01	200
Carol	2018-03-01	400

name	month	amount	total
Alice	2018-01-01	400	1300
Alice	2018-02-01	300	1300
Alice	2018-03-01	600	1300
Bob	2018-01-01	200	750
Bob	2018-02-01	250	750
Bob	2018-03-01	300	750
Carol	2018-01-01	250	850
Carol	2018-02-01	200	850
Carol	2018-03-01	400	850

```
SELECT     *, SUM(amount) OVER W AS total
FROM       Sales
WINDOW     W AS (PARTITION BY name)
ORDER BY name, month
```

Window Functions: Running Total

Sales

name	month	amount
Alice	2018-01-01	400
Alice	2018-02-01	300
Alice	2018-03-01	600
Bob	2018-01-01	200
Bob	2018-02-01	250
Bob	2018-03-01	300
Carol	2018-01-01	250
Carol	2018-02-01	200
Carol	2018-03-01	400

name	month	amount	runningTotal
Alice	2018-01-01	400	400
Alice	2018-02-01	300	700
Alice	2018-03-01	600	1300
Bob	2018-01-01	200	200
Bob	2018-02-01	250	450
Bob	2018-03-01	300	750
Carol	2018-01-01	250	250
Carol	2018-02-01	200	450
Carol	2018-03-01	400	850

```
SELECT    *, SUM(amount) OVER W AS runningtotal
FROM      Sales
WINDOW    W AS (PARTITION BY name ORDER BY month)
ORDER BY  name, month
```

Window Functions: Revenue Growth

Sales

name	month	amount
Alice	2018-01-01	400
Alice	2018-02-01	300
Alice	2018-03-01	600
Bob	2018-01-01	200
Bob	2018-02-01	250
Bob	2018-03-01	300
Carol	2018-01-01	250
Carol	2018-02-01	200
Carol	2018-03-01	400

month	revenue	prevMthRevenue	revenueGrowth
2018-01-01	850	null	null
2018-02-01	750	850	-11.76
2018-03-01	1300	750	73.33

- Revenue = total monthly sales
- Revenue Growth = $\frac{R-R'}{R'} \times 100$
  - R = revenue for a given month
  - R' = revenue for preceding month

Window Functions: Revenue Growth (cont.)

Sales

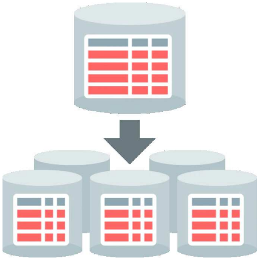
name	month	amount
Alice	2018-01-01	400
Alice	2018-02-01	300
Alice	2018-03-01	600
Bob	2018-01-01	200
Bob	2018-02-01	250
Bob	2018-03-01	300
Carol	2018-01-01	250
Carol	2018-02-01	200
Carol	2018-03-01	400

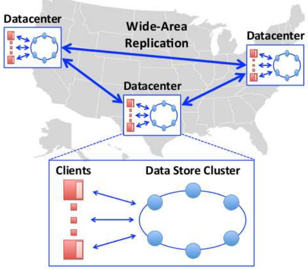
month	revenue	prevMthRevenue	revenueGrowth
2018-01-01	850	null	null
2018-02-01	750	850	-11.76
2018-03-01	1300	750	73.33

```
WITH MthlySales AS
        (SELECT month, SUM(amount) AS revenue
        FROM Sales GROUP BY month),
RevenuePair AS (
        SELECT     month, revenue,
                   lag(revenue) OVER W AS prevMthRevenue
        FROM       MthlySales
        WINDOW     W AS (ORDER BY month))
SELECT      *, ROUND(100.0 * (revenue-prevMthRevenue)/prevMthRevenue,2)
              AS revenueGrowth
FROM        RevenuePair
ORDER BY    month
```

NoSQL Database Systems

- Supports **large-scale data management** challenges of today's web-based applications
  - Database Scalability, High Availability, Low Latency
  - Schema-less data or data with dynamic schema
- Modern **distributed database systems**
  - Data being sharded & replicated across a cluster of servers


Data Sharding (Image: Oracle)



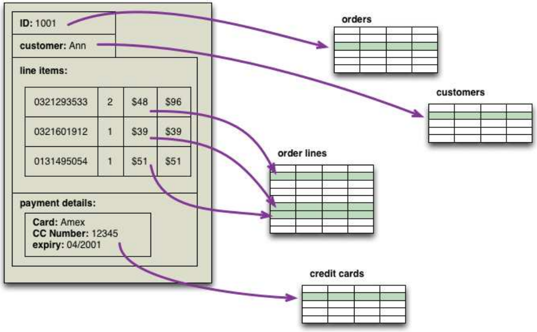
Data Replication (Image: Lloyd, et. al, SOSP 2011)

Early NoSQL Database Systems

- Schema-less data
  - Key-value store
- Simple access API instead of query language
  - Put, Get, Delete
- Limited/No ACID transactional support
- Weak consistency for replicated data
  - Eventual consistency

NoSQL Database Systems

- **Key-value stores** (e.g., Dynamo, Redis)
- **Column-family stores** (e.g., BigTable, Cassandra, HBase)
- **Document stores** (e.g., Cosmos DB, MarkLogic, MongoDB)



(Martin Fowler, 2012)

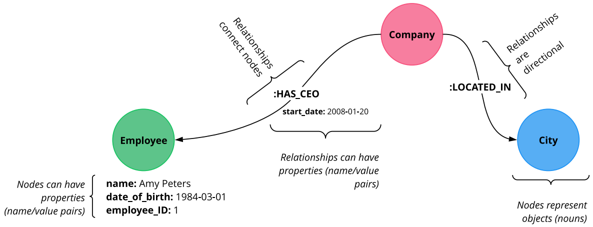
- **Graph database systems** (e.g., JanusGraph, Neo4j)

Graph Database Systems

- Based on different **graph data models**: property graph, RDF graph, hypergraph
- **Property Graphs**
  - Systems: JanusGraph, Neo4j, etc.
- **RDF Graphs**
  - RDF = Resource Description Framework
  - Data stores known as *triplestores* / semantic graph databases
    - Store data as (subject, predicate, object) triples
  - Query language: SPARQL
  - Supports RDF Schema (RDFS) & Web Ontology Language (OWL) inference
  - Uses: Linked Open Data, Knowledge Graphs, etc.
  - Systems: AllegroGraph, GraphDB, etc.
- **Hypergraphs**
  - Systems: HyperGraphDB, Microsoft Graph Engine, etc.

Property Graph Data Model

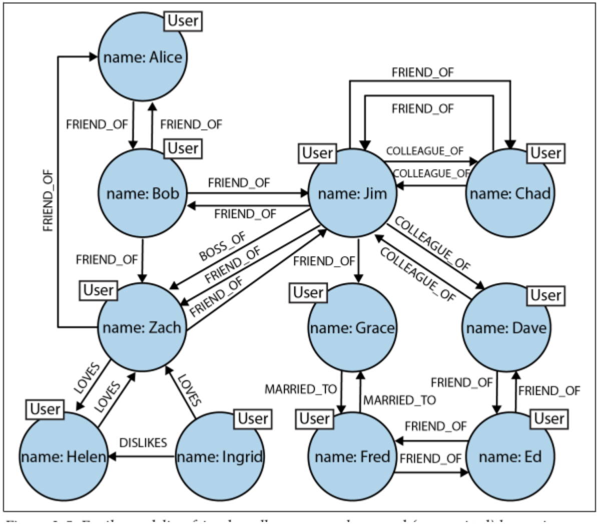
- **Nodes** represent entities
  - Each node has at least one label & possibly properties
- Directed edges represent **relationships** between entities
  - Each relationship has a type & possibly properties
- Each **property** is a key-value pair



Company
Employee
City
Relationships connect nodes
:HAS_CEO
start_date: 2008-01-20
Relationships can have properties (name/value pairs)
:LOCATED_IN
Relationships are directional
Nodes can have properties (name/value pairs)
name: Amy Peters
date_of_birth: 1984-03-01
employee_ID: 1
Nodes represent objects (nouns)

(http://neo4j-contrib.github.io/developer-resources/get-started/graph-database)

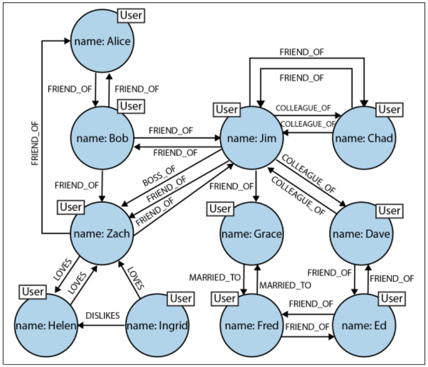
Property Graph Data Model (cont.)



(Robinson, Webber & Eifrem, 2015)

Graph Query Example

Find all users who are friends of Bob that share similar friends as Bob

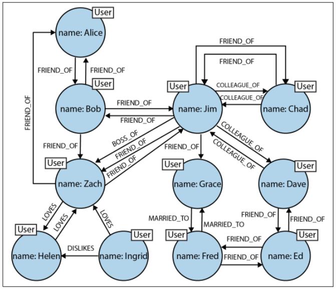


Bob's friends: Alice, Jim, Zach
Alice's friends: Bob
Jim's friends: Bob, Chad, Grace, Zach
Zach's friends: Alice, Jim

Neo4j's Cypher Query Language

- Declarative query language based on property graph model

```
MATCH (e)<-[:FRIEND_OF]-(bob)-[:FRIEND_OF]->(f)-[:FRIEND_OF]->(e)
WHERE bob.name = "Bob"
RETURN f.name AS name,
          count(e) AS score,
          collect(e.name) AS friends
ORDER BY score DESC
```

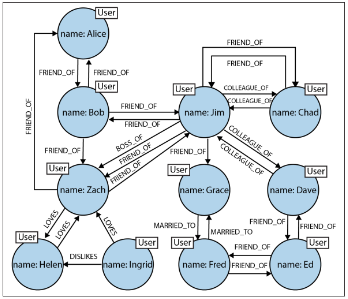


name	score	friends
"Zach"	2	["Alice","Jim"]
"Jim"	1	["Zach"]

Another Cypher Query

For each user, find the number of his/her direct/indirect friends

```
MATCH (u:User)
OPTIONAL MATCH (u)-[:FRIEND_OF*]->(u2:User)
RETURN u.name AS name,
        count(DISTINCT u2) AS numFriends
ORDER BY name
```



name	numFriends
"Alice"	5
"Bob"	5
"Chad"	5
"Dave"	2
"Ed"	2
"Fred"	2
"Grace"	0
"Helen"	0
"Ingrid"	0
"Jim"	5
"Zach"	5

NewSQL Database Systems

- Targeted at OLTP workloads
- Features
  - Relational data model
  - SQL query language
  - ACID transactions
  - Runs on distributed cluster of shared-nothing nodes
- Some examples:
  - Clustrix
  - CockroachDB
  - Google Spanner
  - MemSQL
  - VoltDB

Database-as-a-Service (DBaaS)

- **RDBMS**
  - Amazon RDS (Amazon, Aurora, MySQL, MariaDB, SQL Server, Oracle, PostgreSQL)
    https://aws.amazon.com/rds/
  - Google Cloud SQL (MySQL, PostgreSQL)
    https://cloud.google.com/sql/
- **NoSQL**
  - Amazon DynamoDB
    https://aws.amazon.com/dynamodb/
  - Microsoft Azure Cosmos DB
    https://azure.microsoft.com/en-us/services/cosmos-db/
- **NewSQL**
  - Google Cloud Spanner
    https://cloud.google.com/spanner/