Databases

Query Language

A SQL query returns a multiset: a set that allows duplicates.

Operation	Relational algebra Q	SQL
Base relation	R	`SELECT DISTINCT * FROM R`
Selection	$\sigma_{A>12}(Q)$	`SELECT DISTINCT * FROM R WHERE R.A > 12`
Projection	$\pi_{B,C}(Q)$	`SELECT DISTINCT B, C FROM R`
Rename	$\rho_{B\mapsto E,D\mapsto F}(Q)$	`SELECT DISTINCT A, B as E, C, D as F FROM R`
Union	$Q_1 \cup Q_2$	`SELECT * FROM R UNION SELECT * FROM S`
Intersect	$Q_1 \cap Q_2$	`SELECT * FROM R INTERSECT SELECT * FROM S`
Difference	$Q_1 - Q_2$	`SELECT * FROM R EXCEPT SELECT * FROM S`
Project	$Q_1 \times Q_2$	`SELECT * FROM R CROSS JOIN S`
Natural join	$Q_1 \bowtie Q_2$	`SELECT * FROM R NATURAL JOIN S`

```
-- lookups will now be O(log n)
CREATE INDEX index_name on S(B)
DROP INDEX index_name

SELECT A, B as C FROM table
WHERE condition
GROUP BY field_name -- GROUP BY creates multiple tables
ORDER BY field_name order
LIMIT count

JOIN table ON condition
JOIN table USING (field1, field2, ...)
JOIN table AS alias /* ON/USING ... */

CROSS JOIN table -- cartesian product
LEFT JOIN table  -- keeps all records from the left
                 -- if no matching record, then is NULL
CREATE VIEW name AS /* query */
DROP VIEW name

WITH
  name1 AS (/* query */),
  name2 AS (/* query */)
/* actual query */

INSERT INTO table VALUES (field1, field2, ...)
DELETE FROM table WHERE condition
```

```
MIN(field)
MAX(field)
SUM(field)
AVG(field)
COUNT(field)

-- tri-valued logic
(X = NULL) = NULL
NOT NULL = NULL
TRUE AND NULL = NULL
FALSE OR NULL = NULL

X IS NULL
X IS NOT NULL

-- % same as .* in regex
-- _ same as . in regex
string LIKE "pattern"

-- element in set
element in table
```

```
WITH distances AS
  (SELECT 0 AS n, id2 AS id FROM neighbours
   WHERE id1 = 'source_id' AND id1 = id2
   UNION SELECT n + 1 as n, neighbours.id2 as id FROM distances
         JOIN neighbours ON neighbours.id1 = id
         WHERE NOT (neighbours.id2 IN (SELECT id FROM distances) AND n < 20))
SELECT n, COUNT(*)
FROM (SELECT min(n) AS n, id FROM distances GROUP BY id)
GROUP BY n
```

The ER Model

A **database relation** is the finite set $R \subseteq \{\{(A_1, s_1), (A_2, s_2), ..., (A_n, s_n)\} \mid s_i \in S_i\}$

- A_i is the attribute name
- S_i is the allowed values for A_i

Keyword	Symbol
Entity model things in the real world	Rectangle
Attributes represent properties	Oval
A **key** uniquely identifies an entity instance.	Underline

Cardinality	Symbol
Many to many	
Many to one	
One to one	

A recursive SQL query executes until closure.

- **Transitive closure**: $R \subseteq S \times S$, the transitive closure of R is R^+, the smallest binary relation on S such that $R \subseteq R^+$ and R^+ is transitive.
- $R^+ = \bigcup_n R^n$, if R is finite, there is a k where $R^+ = \bigcup_n^k R^n$

Keys

Keyword	Meaning
Synthetic key	Only have meaning within the database.
Weak entity	Cannot exist without the strong entity existing.
Discriminator	Combined with the key of the strong entity uniquely identifies the weak entity.
Superkey	Z is a superkey of $R(X)$ if $Z \subseteq X$ and for records $u, v : u[Z] = v[Z] \implies u = v$
Key	Z is a key if Z is a superkey and no proper subset of Z is a superkey.
Foreign key	Y is a foreign key for $R(Z, Y)$ pointing to $S(W)$ if $Y \subseteq W$ and $\pi_Y(R) \subseteq \pi_Y(S)$

If Z is a key and Y a foreign key, write $R\left(\underline{Z}, \overline{Y}, X\right)$

Transactions

A **transaction** is a series of atomic queries or changes. If aborted without committing, all changes are reverted and invisible externally.

ACID

Keyword	Definition
Atomicity	All changes appear as a single operation.
Consistency	Every transaction leaves the DB in a consistent state.
Isolation	Transactions appear serialised: intermediate state is invisible to other transactions.
Durability	Once a transaction is completed, changes persists even in a system failure.

BASE

Keyword	Definition
Basically Available	Availability over consistency.
Soft state	Reads may give different values: DB state may change after an update.
Eventual consistency	If there are no updates, the DB will become consistent.

Optimisation

- A **lock** is a hardware/software primitives that provides mutual exclusion.
- A data in DB is **redundant** if it can be reconstructed from other parts of the DB.

A normalised database has no redundant data, all values in a record depends only on the key.

Low Redundancy

- Fewer locks needed, good update throughput
- Used for write-oriented DBs

High Redundancy

- Fewer pointers needed to follow to reach value, good query throughput
- Used for read-oriented DBs

- A read-oriented DB can store snapshots of a write-oriented DB to reduce **lookup cost**.
- Multiple copies of a DB can be stored (**sharding**) to reduce **data movement cost**.

Non-relational DBs

Key-value Store

- **Blobs** are uninterpretable sequence of bytes.
- Values are stored as blobs.

Document Oriented DB

A document oriented DB stores data as **semi-structured objects**.

Key-nestings can be computed for faster searching with particular keys.

Graph Databases

A graph is a collection of nodes and edges.

```
CREATE (node_id:EntityName { /* attributes */ })
CREATE (node_id1) -[:RelationName { /* attributes */ }]-> (node_id2)

MATCH [pattern] RETURN /* attributes */
```

```
MATCH paths=
  allshortestpaths(m: Person { name: 'Kevin Bacon' }
                   -[:ACTED_IN*]- (n:Person))
WHERE m.person_id <> n.person_id
RETURN lengths(paths) / 2 AS bacon_number,
       COUNT(distinct n.person_id) AS total
ORDER BY bacon_number
```

```
(a) --> (b)
(a) -- (b)

// 2 or more incombing edges
() --> (a) <-- ()
(a:Person) --> (b:Movie)

// note: a may equal c
(a) -- (b) -- (c) --> (d)
(a) -[:ACTED_IN] -> (b)

// transitive matching
(a) -[:ACTED_IN*] -> (b)

// all pairs separated by
// 3 to 5 edges
(a) -[*3..5]-> (b)
```