$\pi_{\text{cname,type}}$ account	
cname	type
'McBrien, P.'	'current'
'McBrien, P.'	'deposit'
'Boyd, M.'	'current'
'Poulovassilis, A.'	'current'
'Poulovassilis, A.'	'deposit'
'Bailey, J.'	'current'

π_{type} account
type
'current'
'deposit'

$\pi_{\text{cname,type}}$ account $\div$ π_{type} account

cname
'McBrien, P.'
'Poulovassilis, A.'

π	SELECT
σ	WHERE
$R_1 \times R_2$	FROM R_1, R_2 *or* FROM R_1 CROSS JOIN R_2
$R_1 \bowtie R_2$	FROM R_1 NATURAL JOIN R_2
$R_1 \overset{\theta}{\bowtie} R_2$	FROM R_1 JOIN R_2 ON θ
$R_1 - R_2$	R_1 EXCEPT R_2
$R_1 \cup R_2$	R_1 UNION R_2
$R_1 \cap R_2$	R_1 INTERSECT R_2

If you omit DISTINCT or ALL, then the defaults are:
SELECT ALL
UNION DISTINCT
EXCEPT DISTINCT
INTERSECT DISTINCT

$\sigma_{\text{amount}>1000}$ movement $\cup$ $\sigma_{\text{amount}<-100}$ movement

```
big_movement(Mid, No, Amount, Date) :-
        movement(Mid, No, Amount, Date),
        Amount > 1000.
big_movement(Mid, No, Amount, Date) :-
        movement(Mid, No, Amount, Date),
        Amount < −100.
```

$\sigma_{\text{amount}>1000}$ movement

```
big_credit(Mid, No, Amount, Date) :-
        movement(Mid, No, Amount, Date),
        Amount > 1000.
```

branch $\times$ $\sigma_{\text{rate}>0}$account

```
product_example(BSortcode, BName, Cash, No, Type, CName, Rate, ASortcode) :-
        branch(BSortcode, BName, Cash),
        account(No, Type, CName, Rate, ASortcode),
        Rate > 0.
```

$\pi_{\text{bname,cname}}\sigma_{\text{branch.sortcode=account.sortcode}}$(branch $\times$ account)

```
branch_customers(BName, CName) :-
        branch(Sortcode, BName, _),
        account(_, _, CName, _, Sortcode).
```

Write a query in each of the following languages returning the scheme (name) listing those people that have had at least one child of each gender that appears in the database.

(a) RA

$\pi_{\text{father as name,gender}}\sigma_{\text{IsNotNull(father)}}$ person $\div$ π_{gender} person $\cup$
$\pi_{\text{mother as name,gender}}\sigma_{\text{IsNotNull(mother)}}$ person $\div$ π_{gender} person

```
all_genders(Name) :-
        parent(Name, _),
        ¬missing_gender(Name).
missing_gender(Name) :-
        parent(Name, _),
        person(_, Gender, _, _, _, _, _),
        ¬parent(Name, Gender).
parent(Name, Gender) :-
        person(_, Gender, _, _, Name, _, _),
        isNotNull(Name).
parent(Name, Gender) :-
        person(_, Gender, _, _, _, Name, _),
        isNotNull(Name).
```

```
SELECT name
FROM   person AS parent
WHERE  1=(SELECT COUNT(DISTINCT gender)
          FROM   person
          WHERE  parent.name IN (mother,father))
OR
SELECT mother AS name
FROM   person AS mother
WHERE  mother IS NOT NULL
AND    gender=ALL (SELECT gender
                   FROM   person
                   WHERE  person.mother=mother.mother)
UNION
SELECT father AS name
FROM   person AS father
WHERE  father IS NOT NULL
AND    gender=ALL (SELECT gender
                   FROM   person
                   WHERE  person.father=father.father)
```

Write an SQL query returning the scheme (mid) to find movements that have or might have occurred on 5/1/1999.

```
SELECT   mid
FROM     movement
WHERE    (tdate='5/1/1999') IS NOT FALSE
```

NB will often need to replace IS NOT FALSE by OR tdate IS NULL

```
SELECT person.name,
       person.born_in
FROM   person
       JOIN person AS mother
       ON   person.mother=mother.name
       AND  person.born_in=mother.born_in
```

π_{no} account $-$ π_{no} movement

```
dormant_account(No) :-
        account(No, _, _, _, _),
        ¬movement(_, No, _, _).
```

accounts with all movements less than or equal to 500

```
SELECT no
FROM   account
WHERE  500>=ALL (SELECT amount
                 FROM   movement
                 WHERE  account.no=movement.no)
≡
SELECT no
FROM   account
WHERE  NOT 500<SOME (SELECT amount
                     FROM   movement
                     WHERE  account.no=movement.no)
```

```
SELECT name
FROM   person AS child
WHERE  EXISTS (SELECT *
               FROM   person
               WHERE  child.name IN (mother,father))
OR
SELECT mother AS name
FROM   person AS mother
WHERE  mother IS NOT NULL
UNION
SELECT father AS name
FROM   person AS father
WHERE  father IS NOT NULL
```

List people without a deposit account

```
SELECT DISTINCT cname
FROM   account
WHERE  cname NOT IN
( SELECT cname
  FROM   account
  WHERE  type='deposit')
```

```
SELECT DISTINCT cname
FROM   account
WHERE NOT EXISTS
≡ ( SELECT *
    FROM   account AS deposit_account
    WHERE  type='deposit'
    AND    account.cname=cname)
```

```
SELECT DISTINCT account.*
FROM   account JOIN movement USING (no)
```

Write an SQL query returning the scheme (cname) listing customers that have every type of account that appears in account.

```
SELECT DISTINCT cname
FROM   account AS cust_account
WHERE  NOT EXISTS (SELECT type
                   FROM   account
                   EXCEPT
                   SELECT type
                   FROM   account
                   WHERE  account.cname=cust_account.cname)
```

```
SELECT aname,
       no_aircraft,
       name,
       100.0*no_aircraft/SUM(no_aircraft) OVER (PARTITION BY hq_in) AS pc_country,
       100.0*no_aircraft/SUM(no_aircraft) OVER (PARTITION BY null) AS pc_world
FROM   airline
       JOIN country
       ON   airline.hq_in=country.iso_code
```

SQL Extensions

Pattern Matching: WHERE colum LIKE patten ESCAPE escape_char — % for any char; [A-C] for single A to C; [ABC] for single ABC

Cases: CASE WHEN ~ THEN ~ (ELSE ~ optional) END AS ~,

Modifications: ROUND(n, dp), SUBSTRING(str FROM n1 FOR n2), UPPER(str), CHAR_LENGTH(str), POSITION(char IN str) (must be placed in SELECT clause)

Aggregate: SUM, COUNT, AVG, MIN, MAX excludes NULL

COALESCE(a,b,c) goes down list until non-null

SQL OLAP

GROUP BY: only one output per group, aggregate functions for non grouped colums. NULL is grouped together

HAVING: similar to WHERE but for aggregates (SELECT... FROM...WHERE...GROUP BY.... HAVING)

PARTITION: one output per row, AGG(x) OVER (PARTITION BY y) ....

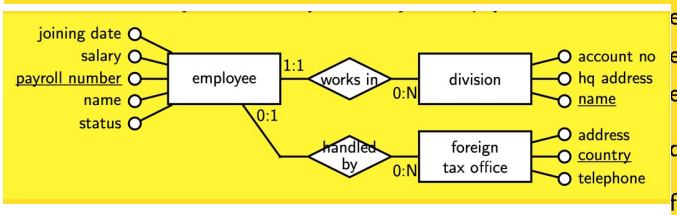
ORDER BY ... DESC: default ascending, RANK() OVER (ORDER BY...) for numbering.

```
SELECT serves.pcode1,
       serves.pcode2,
       COUNT(day) FILTER (WHERE day='Mon') AS monday,
       COUNT(day) FILTER (WHERE day='Tue') AS tuesday
FROM   serves
       NATURAL LEFT JOIN (flight
                          NATURAL JOIN flight_day) AS flight_data
GROUP BY serves.pcode1,
         serves.pcode2
```

```
SELECT acode,
       name
FROM   country
       JOIN airline
       ON   hq_in=iso_code
WHERE  flag_carrier IS NULL
AND    acode NOT IN (SELECT flag_carrier
                     FROM   country
                     WHERE  flag_carrier IS NOT NULL)
```

```
SELECT account.cname,
       COALESCE(SUM(amount) FILTER (WHERE type='current'),0.0)
           AS current_balance,
       COALESCE(SUM(amount) FILTER (WHERE type='deposit'),0.0)
           AS deposit_balance
FROM   account LEFT JOIN movement ON account.no=movement.no
GROUP BY account.cname
```

```
SELECT airline.acode,
       airline.no_aircraft,
       COUNT(country_served.population) AS no_countries,
       SUM(country_served.population) AS total_pop
FROM   airline
       LEFT JOIN (SELECT DISTINCT serves.acode,
                         country.iso_code,
                         country.population
                  FROM   serves
                         JOIN airport
                         ON   airport.pcode IN (serves.pcode1,serves.pcode2)
                         JOIN country USING (iso_code)
                 ) AS country_served
           ON country_served.acode=airline.acode
GROUP BY airline.acode,
         airline.no_aircraft
HAVING SUM(country_served.population)>10000000
```



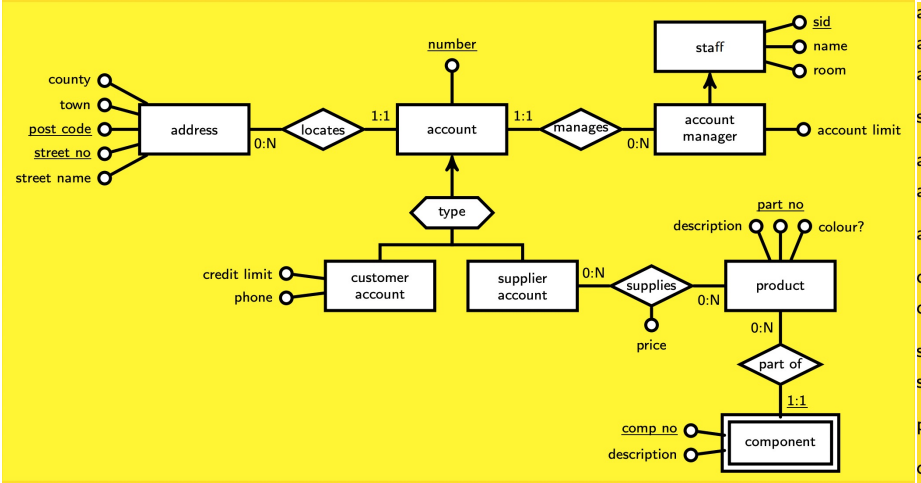
employee(payroll_number,name,status,salary,joining_date,country?)
employee(division_name) $\overset{fk}{\Rightarrow}$ division(name)
employee(country) $\overset{fk}{\Rightarrow}$ foreign_tax_office(country)

division(name,account_no,hq_address)

foreign_tax_office(country,address,telephone)

```
SELECT no,
       SUM(amount) AS balance,
       COUNT(amount) AS no_trans
FROM   movement
GROUP BY no
```

no	balance	no_trans
100	2086.78	3
101	5230.00	2
103	145.50	1
107	245.56	2
119	5600.00	1



account(number,sid,post_code,street_no)
account(sid) $\overset{fk}{\Rightarrow}$ account_manager(sid)
account(post_code,street_no) $\overset{fk}{\Rightarrow}$ address(post_code,street_no)

staff(sid,name,rooom)

account_manager(sid,account_limit)
account_manager(sid) $\overset{fk}{\Rightarrow}$ staff(sid)

address(post_code,street_no,street_name,town,county)

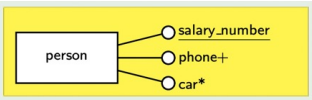
customer_account(number,credit_limit,phone)
customer_account(number) $\overset{fk}{\Rightarrow}$ account(number)

supplier_account(number)
supplier_account(number) $\overset{fk}{\Rightarrow}$ account(number)

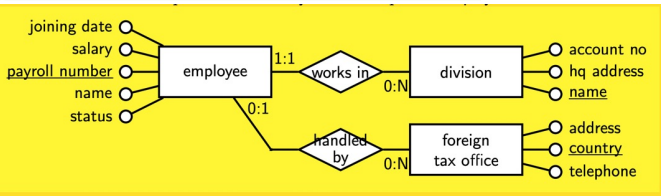
product(part_no,description,colour?)

component(part_no,comp_no,description)
component(part_no) $\overset{fk}{\Rightarrow}$ product(part_no)

supplies(part_no,number,price)
supplies(part_no) $\overset{fk}{\Rightarrow}$ product(part_no)
supplies(number) $\overset{fk}{\Rightarrow}$ supplier_account(number)



person(salary_number)
person_phone(salary_number,phone)
person_phone(salary_number) $\overset{fk}{\Rightarrow}$ person(salary_number)
person_car(salary_number,car)
person_car(salary_number) $\overset{fk}{\Rightarrow}$ person(salary_number)

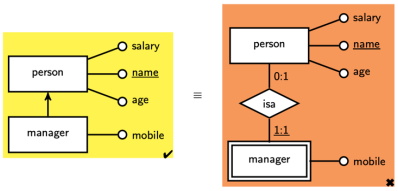


employee(payroll_number,name,status,salary,joining_date,country?)
employee(division_name) $\overset{fk}{\Rightarrow}$ division(name)
employee(country) $\overset{fk}{\Rightarrow}$ foreign_tax_office(country)

division(name,account_no,hq_address)

foreign_tax_office(country,address,telephone)

EER Weak Entity: Must have a key attribute

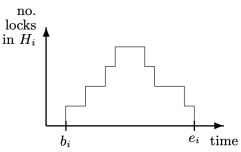


Weak Entities must have at least one key attribute

- A weak entity with one key relationship and no key attributes is equivalent to a subset
- A weak entity with more than one key relationship and no key attributes is eqivalent to a many-many relationship

Two Phase Locking (2PL)

1. read locks $rl[o], \ldots, r[o], \ldots, ru[o]$
2. write locks $wl[o], \ldots, w[o], \ldots, wu[o]$
3. Two phases
   - i. **growing phase**
   - ii. **shrinking phase**
4. refuse $rl_i[o]$ if $wl_j[o]$ already held
   refuse $wl_i[o]$ if $rl_j[o]$ or $wl_j[o]$ already held
5. $rl_i[o]$ or $wl_i[o]$ refused → delay T_i



$\mathcal{A}$	Attributes can be placed on relationships
$\mathcal{D}$	Disjointness between sub-classes can be denoted
$\mathcal{C}$	Look-across cardinality constraints
$\mathcal{H}$	hyper-edges (n-ary relationships) allowed
$\mathcal{L}$	Look-here cardinality constraints
$\mathcal{K}$	Key attributes
$\mathcal{M}$	Mandatory attributes
$\mathcal{N}$	Nested relationships
$\mathcal{O}$	Optional attributes
$\mathcal{S}$	Isa hierarchy between entities
$\mathcal{V}$	Multi-valued attributes
$\mathcal{W}$	Weak entities can be identified

Anomaly	Set	Pattern	Problem
Dirty Write	DW	$w_1[o] \prec w_2[o] \prec e_1$	Sometimes not SR
Dirty Read	DR	$w_1[o] \prec r_2[o] \prec e_1$	Sometimes not RC
Inconsistent Analysis	IA	$r_1[o_a] \prec w_2[o_a],\ w_2[o_b] \prec r_1[o_b]$	Not SR
Lost Update	LU	$r_1[o] \prec w_2[o] \prec w_1[o]$	Not SR

e_i means either c_i or a_i occurring
$op_a \prec op_b$ mean op_a occurs before op_b in a history

conflict

A **conflict** occurs when there is an interaction between two transactions

- $r_x[o]$ and $w_y[o]$ are in H where $x \neq y$
  or
- $w_x[o]$ and $w_y[o]$ are in H where $x \neq y$

Only consider pairs where there is no third operation $rw_z[o]$ between the pair of operations that conflicts with both

Decomposition on an FD

If $R(A_1 \ldots A_n)$ has FD $A_j \rightarrow A_{j+1} \ldots A_n$ then decomposing on the FD to $R_1(A_1 \ldots A_j), R_2(A_j A_{j+1} \ldots A_n)$ is lossless

Keys & FDs

Super-key X of R: X determines all other attributes of R

(non-unique) minimal-key X of R: not possible to remove attributes to form another super-key.

closure S^+: all FDs that can be inferred from S. $S \equiv T$ if $S^+ = T^+$

minimal cover S_c: $S_c^+ = S^+$ and not possible to remove another FD

Normalisation

1st Normal Form (1NF) Every attribute depends on key. (key+ gives all attributes)

Prime Attributes Attributes A of R that are part of a minimal candidate key X of R

2nd Normal Form (2NF) for every non trivial FD $X \rightarrow A$, either X super-key or A prime

Boyce-Codd Normal Form (BCNF) For every non trivial FD $X \rightarrow A$, X is a super key.

Generate minimal keys: 1. Generate minimal cover 2. Find keys where $K^+ = Attr(R)$

Generate minimal cover:
1. rewrite S to have single attribute on RHS of each FD
2. For each FD $X \rightarrow A$, if $(X-B) \rightarrow B$, remove B from X
3. For each FD $X \rightarrow A$, compute X^+ without $X \rightarrow A$. If $A \leq X^+$, remove it

Lossless-join decomposition Decompose R into $R_1, \ldots, R_n$ such that $Attr(R_1) \cup \ldots \cup Attr(R_n) = Attr(R)$

FD-preserving decomposition Lossless decomposition preserve FDs if $S^+ = (S_a \cup S_b)^+$ may need to add more tables on top of lossless

Generating 3/BCNF: (1) Find FD $(X \rightarrow A) \subseteq S$ which violates NF (X not superkey and A non-prime 3NF / X not superkey BCNF) (do possible to preserve FDs)

(2) Decompose R into $R_a(Attr(R)-A)$ and $R_b(XA)$ $(X \subseteq R_a \ \&\ (X \rightarrow A) \subseteq R_b)$

(3) Project S onto new relations and repeat from (1)

Conflict Equivalence

Two histories H_i and H_j are **conflict equivalent** if:

1. Contain the same set of operations
2. Order conflicts (of non-aborted transactions) in the same way.

Conflict Serialisable

a history H is **conflict serialisable** (**CSR**) if $C(H) \equiv_{CE}$ a serial history

Recoverable execution

A **recoverable** (**RC**) history H has no transaction committing before another transaction from which it read

Execution avoiding cascading aborts

A history which **avoids cascading aborts** (**ACA**) does not read from a non-committed transaction

Strict execution

A **strict** (**ST**) history does not read from a non-committed transaction nor write over a non-committed transaction

$ST \subset ACA \subset RC$

A redundant FD $X \rightarrow A$ is equivalent to the transitive or pseudotransitive closure of the other FDs
Since $EG \rightarrow F, F \rightarrow B$
$EG \rightarrow B \Rightarrow \emptyset$

$D^+ = ABCDEFGH$
The only other single attribute on the LHS of an FD is F, and $F^+ = BFH$, so not a candidate key.
Since $AB \rightarrow D$, AB must also be a candidate key.
Since $F \rightarrow B$, AF must also be a candidate key.
Since $EF \rightarrow A$, $F \rightarrow B$, then so must EF be a candidate key.
Since $EG \rightarrow F$, then so must EG be a candidate key.