

**Conditional Expression**

select name, (case when marks >= 70 then ’A’ else ’D’ end) as grade

from Scores;

**Coalesce**

Returns the first non-null value, if all null, returns null value

coalesce(col, [value]) returns value if col is null

**NULLIF**

nullif(v1, v2) returns null if v1 == v2, else return v1

COUNT(\*) / CAST(c.population AS DECIMAL)

-- division by integer gives math floor

select ’Price of ’ || round(price / 1.3) || ’ USD’ as menu

-- CONCAT

INSERT INTO t(column\_list)

VALUES (value\_list),(value\_list), ....;

UPDATE t

SET c1 = new\_value, c2 = new\_value

WHERE condition;

DELETE FROM t; -- delete all from table

DELETE FROM t

WHERE condition;

ALTER TABLE t ADD c;

ALTER TABLE t DROP COLUMN c;

ALTER TABLE t ADD/DROP constraint

**Foreign Key -**

(1) Primary Key Value in reference relation

(2) null value (can have duplicates)

* NO ACTION: rejects delete/update if it violates constraints
* RESTRICT: similar to NO ACTION except that constraint checking can’t be deferred
* CASCADE: propagates delete/update to referencing tuples
* SET DEFAULT: updates foreign keys of referencing tuples to some default value
* SET NULL: updates foreign keys of referencing tuples to NULL

**ER diagrams**

**n-ary relation set:** a relationship with n entity roles. n is the degree of the relationship.

**Total participation constraint** on E wrt R: E participates at least once in relationship R

**Weak entity set:**

1. Does not have its own key
2. must be a many-to-one relationship
3. total participation

|  | Partial Participation constraint  Each instance of E participates in >= 0 instances of R |
| --- | --- |
|  | Key constraint on E wrt R  Each instance of E participates in at more 1 instance of R |
|  | Total participation on E wrt R  Each instance of E participates in at least 1 instance of R |
|  | One-One relationship  Each instance of E participates in exactly 1 instance of R |
|  | E is a weak entity with identifying owner E’ and identifying relationship set R |

**Aggregation**

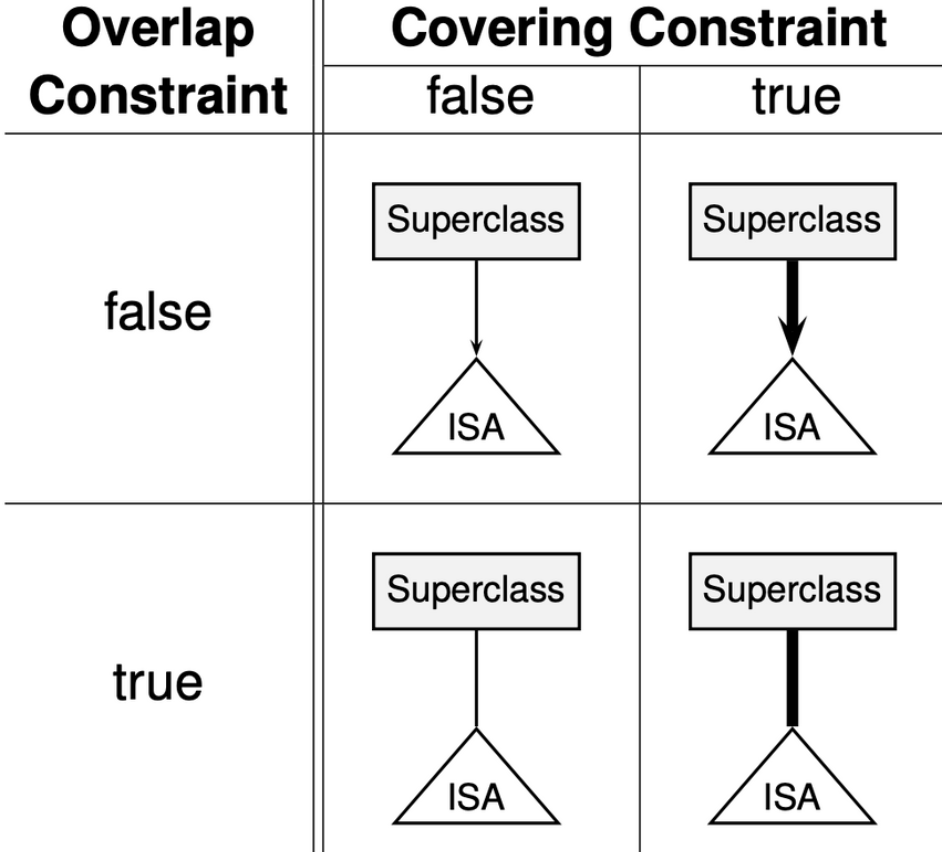
The box connected to the relationship denotes that it is participating in the other relationship as an aggregation.

Conversion: If relationship R is the entity in aggregation then must be defined first.

**ISA Hierarchies**

**Overlap constraint:** Entity can belong to multiple classes

**Covering constraint:** Entity must belong to >= 1 class



Conversion: REFERENCES PARENT ON DELETE CASCADE

**Function**

| /\* if else \*/  IF temp > 60 THEN  temp := temp/2;  ELSIF temp > 50 THEN  temp := temp-20;  ELSE  temp := temp - 10;  END IF; | /\* while loop \*/  WHILE temp > 30 LOOP  temp := temp / 2;  END LOOP; |
| --- | --- |

FOREACH d IN ARRAY denoms LOOP

temp := temp / d;

END LOOP;

RETURN QUERY SELECT mark1, mark2;

-- can just select value into mark and no need return if not return table table

SELECT \* FROM fname(1, 2);

SELECT fname(1, 2); -- both returns table

**Procedure**

CREATE OR REPLACE PROCEDURE AddGradeAttr() AS $$

ALTER TABLE Scores

ADD COLUMN IF NOT EXISTS Grade CHAR(1) DEFAULT NULL;

SELECT \* FROM Scores;

$$ LANGUAGE sql;

CALL AddGradeAttr();

**Cursor**

* FETCH curs INTO r;
* FETCH NEXT FROM curs INTO r;
* FETCH PRIOR FROM curs INTO r; -- Fetch from previous row
* FETCH FIRST FROM curs INTO r;
* FETCH LAST FROM curs INTO r;
* FETCH ABSOLUTE 3 FROM curs INTO r;
* FETCH RELATIVE -2 FROM curs INTO r;
* FETCH [PRIOR | FIRST | LAST | ABSOLUTE n | RELATIVE n] [FROM] <cursor> INTO <var>
* MOVE [PRIOR | FIRST | LAST | ABSOLUTE n | RELATIVE n] [FROM] <cursor>; [UPDATE | DELETE] <table> ... WHERE CURRENT OF curs;

**Trigger**

* TG\_OP: trigger operation
* TG\_TABLE\_NAME: trigger table name

CREATE CONSTRAINT TRIGGER trigger\_name AFTER INSERT ON table\_name

DEFERRABLE INITIALLY DEFERRED

FOR EACH ROW EXECUTE FUNCTION func();

-- for constraint that is DEFERRABLE INITIALLY IMMEDIATE

BEGIN TRANSACTION;

SET CONSTRAINTS bal\_check\_trigger DEFERRED; -- defer

COMMIT;

| BEFORE INSERT/UPDATE/DELETE | Return value affects the action   * RETURN NULL, no action performed; * RETURN OLD is same as RETURN NULL for INSERT if OLD is not initialised   AFTER: Return value does not matter |
| --- | --- |
| INSTEAD OF INSERT/UPDATE/DELETE | * Occurs in place of the specified action (only applicable for views) * only allowed on row‐level * RETURN NULL will cause all operations (including other triggers) to be ignored; * Returning non-null value means proceed as normal |
| FOR EACH ROW | * Calls the trigger function for each tuple involved in the statement |
| FOR EACH STATEMENT | * Calls the trigger function once for the whole statement * Return value does not matter * RETURN NULL would not make the database omit the subsequent operations * For subsequent operations to be omitted, raise exception * NEW and OLD not defined |
| * No SELECT in WHEN * No OLD in WHEN for INSERT * No NEW in WHEN for DELETE * No WHEN for INSTEAD OF   CREATE TRIGGER for\_Elise\_trigger BEFORE INSERT ON Scores FOR EACH ROW WHEN (NEW.Name = 'Elise') EXECUTE FUNCTION for\_Elise\_func();  CREATE OR REPLACE FUNCTION for\_Elise\_func() RETURNS TRIGGER AS $$ BEGIN  NEW.Mark := 100;  RETURN NEW; END; $$ LANGUAGE plpgsql; | |

**Functional Dependencies**

Axiom of Reflexivity: ABCD → ABC

Axiom of Augmentation: if A → B, then AC → BC

Axiom of Transitivity: If A → B and B → C then A → C

Rule of Decomposition: If A → BC, then A → B and A → C

Rule of Union: If A → B and A → C, then A → BC

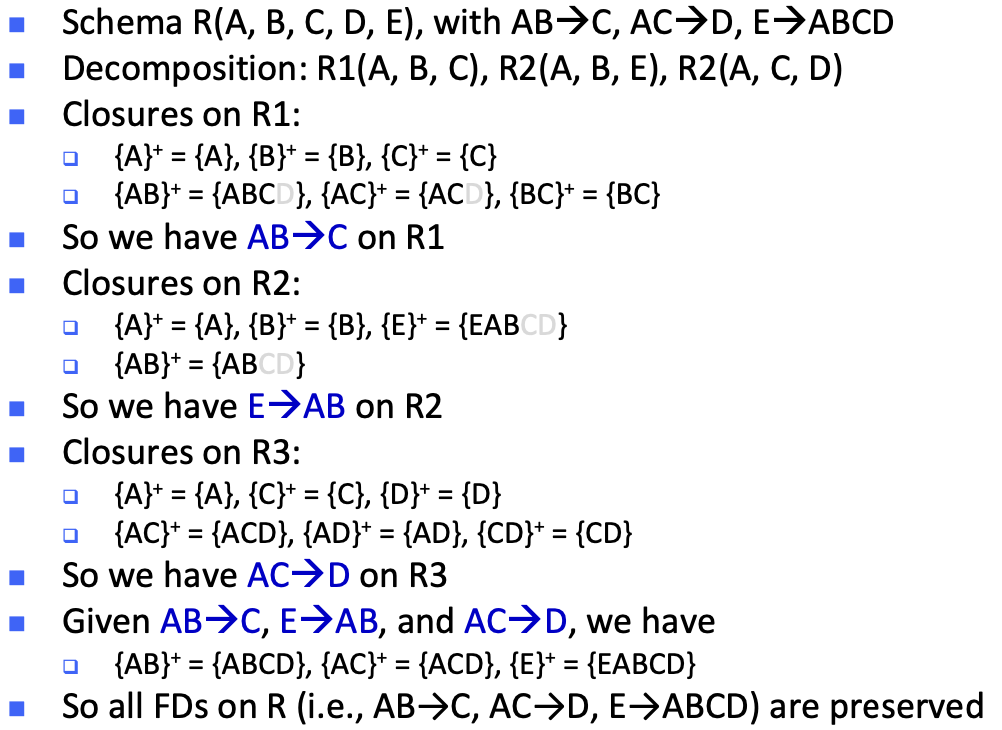
Prime attributes: attribute that appears in a key

**Lossless-join Decomposition:** common attribute in R1 and R2 constitute a superkey of R1 or R2

**Dependency Preservation**

S - original set. S’ - decomposed set of fd.

S and S’ must be equivalent so every FD in S can be derived from S’ and every FD in S’ from S.



| **BCNF** | **3NF** |
| --- | --- |
| if every non‐trivial and decomposed FD’s LHS is a superkey. | iff for every non‐trivial and decomposed FD  - Either LHS is a superkey  - Or the RHS is a prime attribute (i.e., it appears in a key) |
| violation of BCNF, iff we have a closure that satisfies the "more but not all" condition | Violation of 3NF, iff we have a closure that satisfies the “more but not all” property and the extra attribute is not a prime attribute |

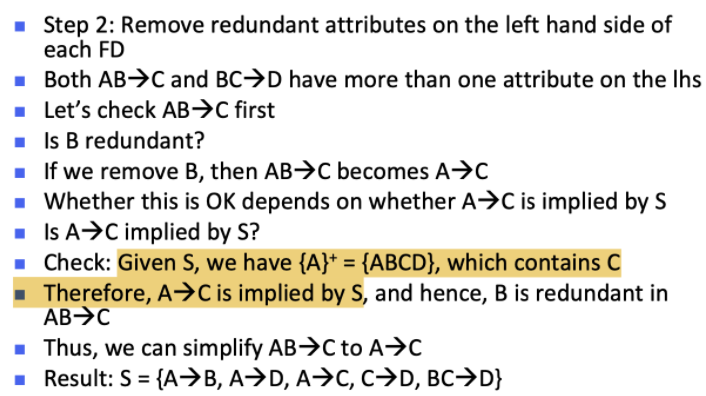
**BCNF Decomposition Algo**

* recursively decompose into table with attributes in the closure {X}+ that violates BCNF and table with X and other attributes not in that closure.
* If we have a table R(X, Y, Z) with {X}+ = {X, Y}, then decompose R into R1(X, Y) and R2(X, Z) until all tables are in BCNF

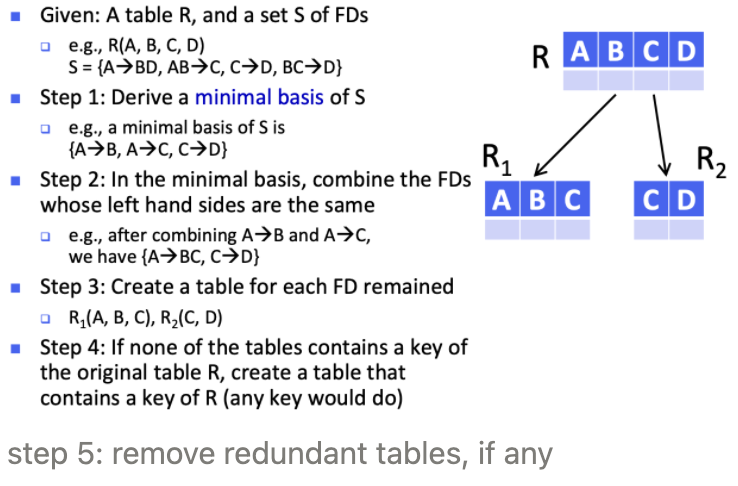
**Minimal Basis Algo**

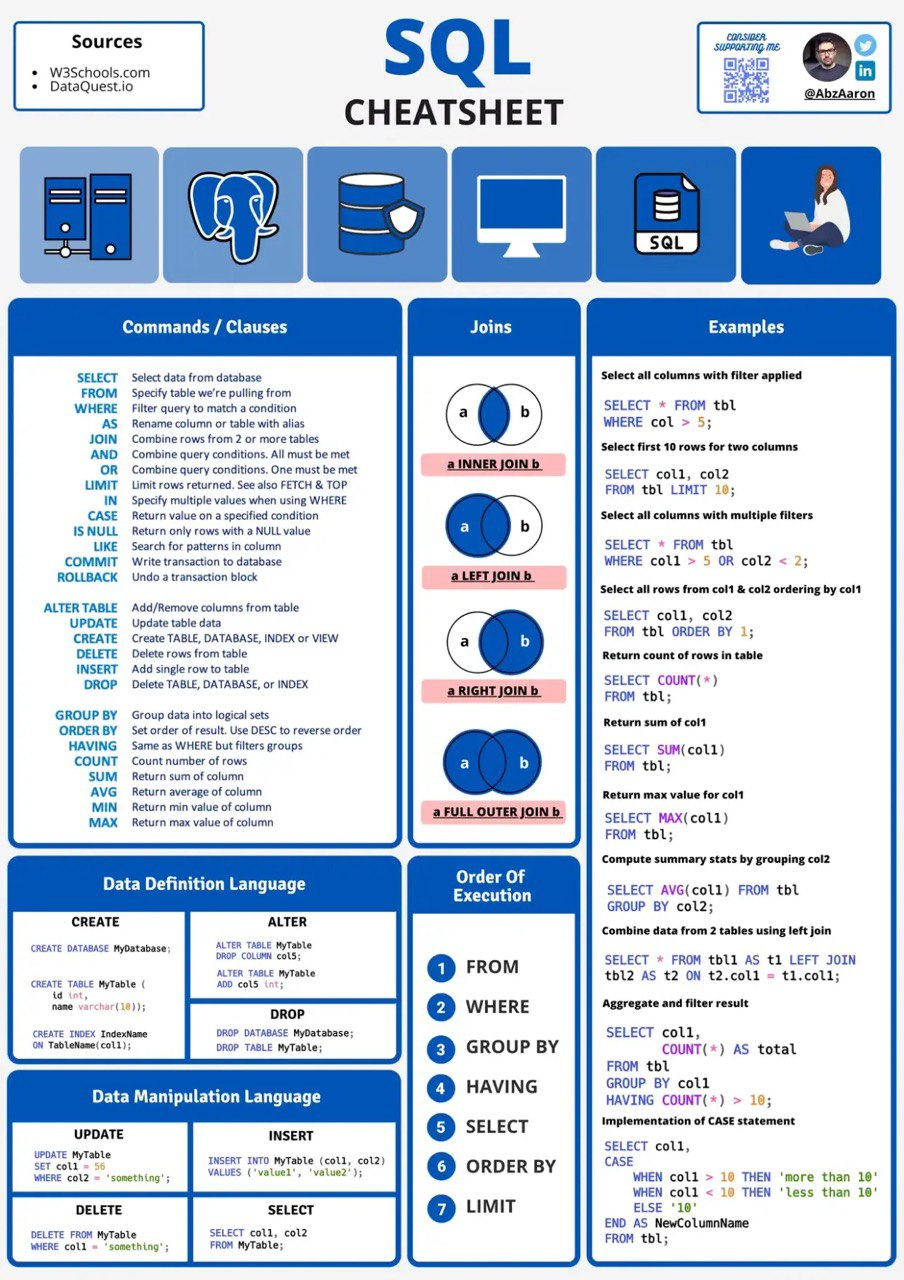
1. Transform the FDs, so that each RHS contains only one attribute
2. Remove redundant attributes on the LHS of each FD
3. Remove redundant FDs

E.g. S = {A → BD, AB → C, C → D, BC → D}



**3NF Decomposition Algo**





AY21/22 Semester 2

Ang Koon Hwee, Wu Xiao Yun

-- e.g. first 3 consecuvtive coins that are down by more than 5%

CREATE OR REPLACE FUNCTION consCryptosDown (IN n INT)

RETURNS TABLE(rank INT, sym CHAR(4)) AS $$ DECLARE

curs CURSOR FOR (SELECT \* FROM cryptosRank WHERE changes < -5);

r1 RECORD;

r2 RECORD;

-- assignment e.g. temp INT := 0;

BEGIN

OPEN curs;

LOOP

FETCH curs INTO r1;

EXIT WHEN NOT FOUND;

FETCH RELATIVE (n-1) FROM curs INTO r2;

EXIT WHEN NOT FOUND;

IF r2.rank - r1.rank = n-1 THEN

MOVE RELATIVE -(n) FROM curs;

FOR c IN 1..n LOOP FETCH curs INTO r1;

rank := r1.rank;

sym := r1.symbol;

RETURN NEXT;

END LOOP;

CLOSE curs; RETURN;

END IF;

MOVE RELATIVE -(n-1) FROM curs;

END LOOP; CLOSE curs;

END; $$ LANGUAGE plpgsql;