CS2102 Part 2

L7: PL/pgSQL

Statement Level Interface

1. write a program that mixes host language with SQL.
2. preprocess the program using a preprocessor.
3. compile the program into an executable code.

```
void main() {
```

`EXEC SQL BEGIN DECLARE SECTION;`
`  char name[30]; int mark;`
`EXEC SQL END DECLARE SECTION;`
	Declaration

`EXEC SQL CONNECT @localhost USER john;`	Connection
`// some code that assigns values to` `// name and mark.`	Host language
`EXEC SQL INSERT INTO` `  Scores (Name, Mark) VALUES (:name, :mark);`	Query execution
`EXEC SQL DISCONNECT;`	Disconnect

```
}
```

Fixed SQL query, i.e. static SQL

```
void main() {
    EXEC SQL BEGIN DECLARE SECTION;
      char *query; char name[30]; int mark;
    EXEC SQL END DECLARE SECTION;

    EXEC SQL CONNECT @localhost USER john;

    // some code that assigns values to
    // name and mark

    // assign any SQL statement to the query,
    // the query may include name and/or mark.

    EXEC SQL EXECUTE IMMEDIATE :query;

    EXEC SQL DISCONNECT;
}
```

Declaration

Connection

Host language

Query execution

Disconnect

Dynamic SQL generates queries at runtime

Call Level Interface

1. write in host language only
    - Need to load a library that provides APIs to access the DB, e.g., libpq, psqlODBC, JDBC, ODBC, etc.
2. compile the program into an executable code

```
void main() {
    char *query; char name[30]; int mark;

    connection C("dbname = testdb user = postgres \
      password = test hostaddr = 127.0.0.1 \
      port = 5432");

    // assign any SQL statement to the query,
    // the query may include name and/or mark.

    work W(C);
    W.exec(query);
    W.commit();

    C.disconnect();
}
```

Declaration

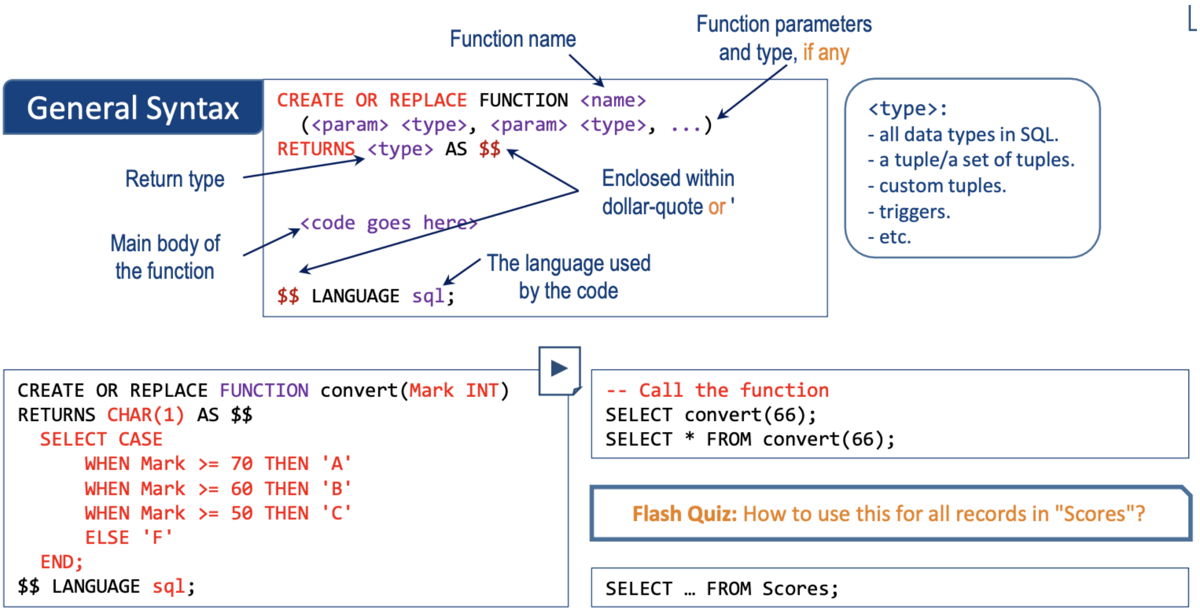
Connection

Query execution

Disconnect

Functions

- returns a value




```
-- BASIC EXAMPLE USAGE
SELECT Name, convert(Mark) FROM Scores;

SELECT Name
FROM Scores WHERE convert(Mark) = 'B';

-- RETURN A TUPLE
CREATE OR REPLACE FUNCTION GradeStudent (Grade CHAR(1))
RETURNS Scores AS $$

SELECT *
FROM Scores
WHERE convert(Mark) = Grade;

$$ LANGUAGE sql;

-- RETURNS A SET OF TUPLES
CREATE OR REPLACE FUNCTION GradeStudents (Grade CHAR(1))
RETURNS SETOF Scores AS $$ ...
$$ LANGUAGE sql;

-- RETURNS A SET OF CUSTOMISED TUPLES
CREATE OR REPLACE FUNCTION GradeStudents (Grade CHAR(1))
RETURNS SETOF RECORD AS $$ ...
$$ LANGUAGE sql;

-- SIMPLIFY PARAMS FOR CUSTOM TUPLES
CREATE OR REPLACE FUNCTION CountGradeStudents()
RETURNS TABLE(MARK CHAR(1), COUNT INT) AS $$
SELECT convert(Mark), count(*)
FROM scores
GROUP BY convert(Mark);
```

```
$$ LANGUAGE sql;

SELECT CountGradeStudents();

-- RETURNS VOID
CREATE OR REPLACE FUNCTION AddGradeAttr()
RETURNS VOID AS $$

  ALTER TABLE Scores
  ADD COLUMN IF NOT EXISTS Grade CHAR(1) DEFAULT NULL;
  UPDATE Scores SET Grade = convert(Mark);

  SELECT * FROM Scores;

$$ LANGUAGE sql;

SELECT AddGradeAttr();
```

Variables & Control Flows

```
CREATE OR REPLACE FUNCTION splitMarks
(IN name1 VARCHAR(20), IN name2 VARCHAR(20))
RETURNS TABLE(Mark1 INT, Mark2 INT)
AS $$
DECLARE
  temp INT := 0;
BEGIN
  /* two ways of assignment */
  SELECT mark INTO mark1 FROM Scores WHERE name = name1;
  SELECT mark INTO mark2 FROM Scores WHERE name = name2;
  temp := (mark1 + mark2) / 2;

  /* 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;

  /* while (true) { if (temp < 30) break; ... } */
  LOOP
    EXIT WHEN temp < 30;
    temp := temp / 2;
  END LOOP;

  FOREACH d IN ARRAY denoms LOOP
    temp := temp / d;
```

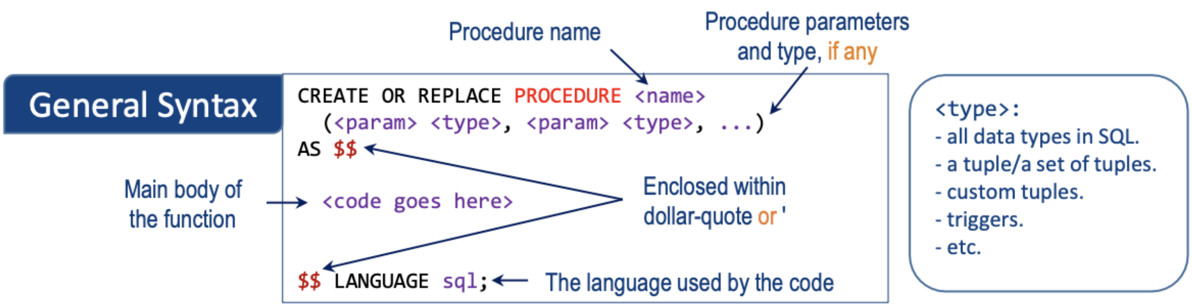
```
  END LOOP;

  UPDATE Scores
  SET mark = temp
  WHERE name = name1 OR name = name2;

  RETURN QUERY SELECT mark1, mark2;
END;
$$ LANGUAGE plpgsql;
```

Procedures

- no return value




```
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

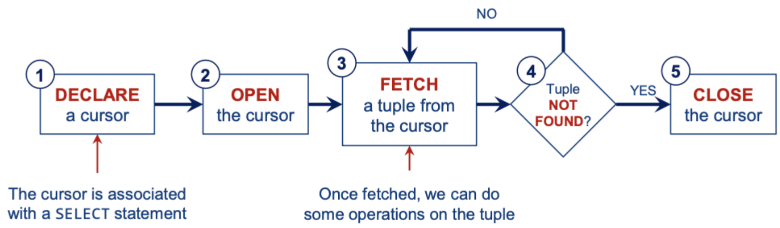
- access each individual row returned by a SELECT statement

```
-- Cursor movement
FETCH curs INTO r;
FETCH NEXT FROM curs INTO r;

-- other variants
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 3rd tuple
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;
```



Rank	Symbol	Changes
1	BTC	-6%
3	DOGE	-6%
4	ZIL	-7%
5	XMR	-8%
6	SHIB	-8%
8	LTC	-7%
9	XRP	-7%
10	BNB	-6%

```
-- 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;
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;
```

L8: Trigger

Trigger Timing, Return Values

- `BEFORE INSERT/UPDATE/DELETE`
    - Occurs before the action has modified the database
    - Return value affects the action (if `RETURN NULL`, no action performed; `RETURN OLD` is same as `RETURN NULL` for INSERT if `OLD` is not initialised)

```
CREATE TRIGGER for_Elise_trigger
BEFORE INSERT ON Scores
FOR EACH ROW EXECUTE FUNCTION for_Elise_func();

CREATE OR REPLACE FUNCTION for_Elise_func()
RETURNS TRIGGER AS $$
BEGIN
  IF (NEW.Name = 'Elise') THEN NEW.Mark := 100;
  END IF;
  RETURN OLD; -- same as RETURN NULL
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION for_Elise_func()
RETURNS TRIGGER AS $$ BEGIN
  OLD.Name := 'Haha';
  OLD.Mark := 0;
  RETURN OLD; -- ('Haha', 0) will be inserted
END;
$$ LANGUAGE plpgsql;
```

- `AFTER INSERT/UPDATE/DELETE`
    - Occurs after the action has modified the database
    - 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
- Returning `NULL` will cause all operations (including other triggers) to be ignored;
- Returning non-null value means proceed as normal

```
CREATE TRIGGER update_max_trigger
INSTEAD OF UPDATE ON Max_Score
FOR EACH ROW EXECUTE FUNCTION update_max_func();

CREATE OR REPLACE FUNCTION update_max_func()
RETURNS TRIGGER AS $$
BEGIN
  UPDATE Scores
  SET Mark = NEW.Mark
  WHERE Name = OLD.Name;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger Level

- `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
  - Note: at statement level, return value does not matter
  - `RETURN NULL` would not make the database omit the subsequent operations
  - For subsequent operations to be omitted, raise exception
- Caveats
  - `INSTEAD OF` is only allowed on the row level
  - `NEW` and `OLD` are not defined for the statement level

Trigger Condition

Move conditionals to the trigger rather than the trigger function

- No `SELECT` in `WHEN`

- No `OLD` in WHEN for `INSERT`
- No `NEW` in WHEN for `DELETE`
- No `WHEN` for `INSTEAD OF`

```
CREATE TRIGGER trigger
BEFORE INSERT ON table
FOR EACH ROW
WHEN (condition)
EXECUTE FUNCTION func();

-- e.g.
CREATE TRIGGER for_Elise_trigger
BEFORE INSERT ON Scores
FOR EACH ROW EXECUTE FUNCTION for_Elise_func();

CREATE OR REPLACE FUNCTION for_Elise_func()
RETURNS TRIGGER AS $$
BEGIN
  IF (NEW.Name = 'Elise') THEN
    NEW.Mark := 100;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- this trigger function only cares about the case when NEW.Name = 'Elise' => too many
checks if most entries are not 'Elise'

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;
```

Deferred Triggers

- Only works with AFTER and FOR EACH ROW

```
CREATE CONSTRAINT TRIGGER trigger_name
AFTER INSERT ON table_name
DEFERRABLE INITIALLY DEFERRED
FOR EACH ROW
EXECUTE FUNCTION func();

BEGIN TRANSACTION;
...
COMMIT; -- trigger activated

CREATE CONSTRAINT TRIGGER bal_check_trigger
AFTER INSERT OR UPDATE OR DELETE ON Account
DEFERRABLE INITIALLY IMMEDIATE -- trigger is not deferred by default
FOR EACH ROW
EXECUTE FUNCTION bal_check_func();

BEGIN TRANSACTION;
SET CONSTRAINTS bal_check_trigger DEFERRED; -- defer
UPDATE Account
SET Bal = Bal – 100
WHERE AID = 1;

UPDATE Account
SET Bal = Bal + 100
WHERE AID = 2;
COMMIT;
```

Multiple Triggers - Order of Triggers

- Order based on trigger timing and level:
  - BEFORE statement-level trigger
  - BEFORE row-level triggers
  - AFTER row-level triggers
  - AFTER statement-level triggers
- Within each type:
  - Alphabetical order
  - Output of the previous trigger could affect the next
  - If previous (BEFORE) trigger returns NULL: subsequent triggers will not run

```
/* BREAK INTO CASES DEPENDING ON TABLE OPERATOR */
CREATE OR REPLACE FUNCTION scores_log2_func()
```

```
RETURNS TRIGGER AS $$
BEGIN
  IF (TG_OP = 'INSERT') THEN
    INSERT INTO Scores_Log2 SELECT NEW.Name, 'Insert', CURRENT_DATE;
    RETURN NEW;
  ELSEIF (TG_OP = 'DELETE') THEN
    INSERT INTO Scores_Log2 SELECT OLD.Name, 'Delete', CURRENT_DATE;
    RETURN OLD;
  ELSEIF (TG_OP = 'UPDATE') THEN
    INSERT INTO Scores_Log2 SELECT NEW.Name, 'Update', CURRENT_DATE;
    RETURN NEW;
  END IF;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER scores_log2_trigger
AFTER INSERT OR DELETE OR UPDATE ON Scores
FOR EACH ROW EXECUTE FUNCTION scores_log2_func();

-- TG_TABLE_NAME: the name of table that caused the trigger invocation
```

L9: Functional Dependencies

Normal Forms

- def: minimum requirements to reduce data redundancy and improve data integrity

Name	<u>NRIC</u>	<u>PhoneNumber</u>	HomeAddress
Alice	1234	67899876	Jurong East
Alice	1234	83848384	Jurong East
Bob	5678	98765432	Pasir Ris

redundancy: ALICE is stored twice

Update Anomalies

- update one record of ALICE but forgot to update the other

Delete Anomalies

- e.g. Bob no longer uses a phone but cannot remove his phone number (phone num is primary key attribute which cannot be NULL)

Insertion Anomalies

- e.g. wants to insert `Name = Cathy, NRIC = 9394, HomeAddress = YiShun`
- no phone number, cannot insert

Functional Dependencies Basics

def: $A_1A_2...A_m \rightarrow B_1B_2...B_n$ if whenever two objects have the same values on $A_1A_2...A_m$, they always have the same values on $B_1B_2...B_n$

- similar to function definition in math
- e.g. NRIC → Name: if two tuples have the same NRIC value, then they have the same Name

Example: Purchase(CustomerID, ProductID, ShopID, Price, Date)

- requirement: no two customers buy the same product
  - ProductID → CustomerID
- requirement: No two shops sell the same product on the same date
  - ProductID, Date → ShopID
- requirement: No shop should sell the same product to the same customer on the same date at two different prices
  - CustomerID, ProductID, ShopID, Date → Price

Armstrong's Axioms

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

Additional Rules

Rule of Decomposition

- If A → BC, then A → B and A → C

<u>Rule of Union</u>

- If A → B and A → C, then A → BC

Example working:

1.	a→b	(given)
2.	c→d	(given)
3.	ac→bc	(augmentation of (1) with c)
4.	bc→bd	(augmentation of (2) with b)
5.	ac→bd	(transitivity of (2) and (3))

Closure

- Let $S = A_1, A_2, ..., A_3$ be a set of attributes
- The closure of S is the set of attributes that can be decided by $A_1, A_2, ..., A_n$
    1. Initialise the closure to $\{A_1, A_2, ..., A_3\}$
    2. If there is an FD: $A_i, A_j, ...A_m \rightarrow B$, s.t. $A_i, A_j, ...A_m$ are all in the closure, then put B into the closure
    3. Repeat step 2 until we cannot find any new attribute to put into the closure
- Notation: $\{A_1, A_2, ..., A_k\}^+$
- e.g. Given A → B, B → C, C → D, D → E
    - {A}+ = {A,B,C,D,E}
    - {B}+ = {B,C,D,E}
    - {D}+ = {D, E}
    - {E}+ = {E}
- To prove that X → Y holds, we only need to show that {X}+ contains Y

Superkeys

Definition: a set of attributes in a table that decides all other attributes

Keys

Definition: minimal superkey, i.e. if we remove any attribute from the superkey, it will not be a superkey anymore

Algorithm for finding keys:

- Consider every subset of attributes in T:
  - A, B, C, …, AB, BC, CA, …, ABC, …
- Derive the closure of each subset:
  - $\{A\}^+$, $\{B\}^+$, $\{C\}^+$, …, $\{AB\}^+$, $\{BC\}^+$, $\{AC\}^+$, …, $\{ABC\}^+$, …
- Identify all superkeys based on the closures
- Identify all keys from the superkeys

- A table R(A, B, C), with $A \rightarrow B$, $B \rightarrow C$
- Steps for finding keys:
  - Consider every subset of attributes in T:
    - A, B, C, AB, BC, CA, ABC
  - Derive the closure of each subset:
    - $\{A\}^+=\{ABC\}$, $\{B\}^+=\{BC\}$, $\{C\}^+=\{C\}$
    - $\{AB\}^+=\{ABC\}$, $\{BC\}^+=\{BC\}$, $\{AC\}^+=\{ABC\}$, $\{ABC\}^+=\{ABC\}$
  - Identify all superkeys based on the closures
    - A, AB, AC, ABC
  - Identify all keys from the superkeys
    - A

Tips on finding keys

1. Check the smallest attribute sets first
2. Find attributes that do not appear in the RHS of any FD $\Rightarrow$ these attributes must be in every key

- A table R(A, B, C, D, E)
- $AB \rightarrow C$, $C \rightarrow B$, $BC \rightarrow D$, $CD \rightarrow E$
- A must be in every key
- Compute the closures:
  - $\{A\}^+ = \{A\}$
  - $\{AB\}^+ = \{ABCDE\}$
  - $\{AC\}^+ = \{ACBDE\}$
  - $\{AD\}^+ = \{AD\}$, $\{AE\}^+ = \{AE\}$
  - $\{ADE\}^+ = \{ADE\}$
- Keys: AB, AC

Prime Attributes

- Attributes that appear in a key

L10: Boyce-Codd Normal Form (BCNF)

- if every non-trivial and decomposed FD has a superkey as its LHS

Non-trivial and Decomposed FD

- non-trivial: attribute does not appear on both LHS and RHS
- decomposed: RHS has only one attribute
  - BC → DE: BC → D and BC → E

<u>Derivation</u>

1. Compute the closure of all subsets
2. For each closure, remove the trivial attributes
3. Derive non-trivial and decomposed FDs from each closure

e.g. R(A, B, C) with A → B, B → A, B → C given

$\{A\}^+ = \{ABC\}$,	$\{B\}^+ = \{ABC\}$,	$\{C\}^+ = \{C\}$	A→B,	A→C,	B→A,	B→C
$\{AB\}^+ = \{ABC\}$,	$\{AC\}^+ = \{ABC\}$,	$\{BC\}^+ = \{ABC\}$	AB→C,	AC→B,	BC→A	

- Key: A, B
- for each of the above FD, the LHS is a superkey, hence R satisfies/is in BCNF

BCNF Check

1. Compute the closure of each attribute subset
2. Derive the keys of R (using closures)
3. Derive all non-trivial and decomposed FDs on R (using closures)
4. Check the non-trivial and decomposed FDs to see if they satisfy the BCNF requirement
5. If all of them satisfy the requirement, then R is in BCNF

R(A, B, C, D) with FDs AB → C, C → D, and D→A

2. Derive the keys of R: AB, BC, BD
3. Derive the non-trivial and decomposed FDs on R

- $\{A\}^+ = \{A\}$, $\{B\}^+ = \{B\}$, $\{C\}^+ = \{ACD\}$, $\{D\}^+ = \{AD\}$
- $\{AB\}^+ = \{ABCD\}$, $\{AC\}^+ = \{ACD\}$, $\{AD\}^+ = \{AD\}$
- $\{BC\}^+ = \{ABCD\}$, $\{BD\}^+ = \{ABCD\}$, $\{CD\}^+ = \{ACD\}$
- $\{ABC\}^+ = \{ABD\}^+ = \{BCD\}^+ = \{ABCD\}$
- $\{ACD\}^+ = \{ACD\}$
- $\{ABCD\}^+ = \{ABCD\}$

R(A, B, C, D) with FDs AB → C, C → D, and D→A

2. Derive the keys of R: AB, BC, BD
3. Derive the non-trivial and decomposed FDs on R

- C→A, C→D, D→A
- AB→C, AB→D, AC→D
- BC→A, BC→D, BD→A, BD→C, CD→A
- ABC→D, ABD→C, BCD→A

4. For each non-trivial and decomposed FD, check whether its left hand side is a super-key

Not in BCNF

Simplified BNCF Check

- we have a violation of BCNF, iff we have a closure that satisfies the "more but not all" condition

1. Compute the closure of each attribute subset
2. Check if there is a closure $\{A_1, A_2, ..., A_k\}^+$ such that
   - the closure contains some attribute not in $\{A_1, A_2, ..., A_k\}$
   - the closure does not contain all attributes in the table
   - i.e. “more but not all” closure
   - if such a closure exists, then R is NOT in BCNF

R(A, B, C, D) with FDs B → C, B → D

- Compute the closure of each attribute subset
  - $\{A\}^+ = \{A\}$, $\{B\}^+ = \{BCD\}$, $\{C\}^+ = \{C\}$, $\{D\}^+ = \{D\}$
- $\{B\}^+ = \{BCD\}$ stratifies the "more but not all" property
- So it indicates a violation of BCNF

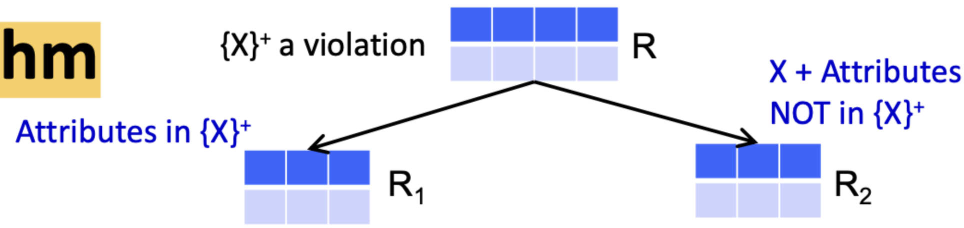
R(A, B, C, D) with FDs A → B, B → C, C→D, and D→A

- Compute the closure for each subset of the attributes in R
  - $\{A\}^+ = \{ABCD\}$, $\{B\}^+ = \{ABCD\}$, $\{C\}^+ = \{ABCD\}$, $\{D\}^+ = \{ABCD\}$
  - The other closures are all {ABCD}
- There is no closure satisfying the "more but not all" property
- So there is no violation of BCNF

Decomposition (Normalisation)

- If a table has only two attributes, then it must be in BCNF
  - choose combination s.t. it results in fewer attributes during decomposition

Algorithm



- Input: a table R

1. Find a subset X of the attributes in R, such that its closure $\{X\}^+$ (i) contains more attributes than X does, but (ii) does not contain all attributes in R
2. Decompose R into two tables R_1 and R_2, such that
   - R_1 contains all attributes in $\{X\}^+$
   - R_2 contains all attributes in X as well as the attributes not in $\{X\}^+$
3. If R_1 is not in BCNF, further decompose R_1;
   If R_2 is not in BCNF, further decompose R_2

Decomposition with Implicit FD

Given: R(A, B, C, D, E) and A → B, BC → D

1. {A}+ = {A, B} violates the “more but not all” property.
2. Decompose into R1(A, B) and R2(A, C, D, E)
   a. R1 is in BCNF
3. Derive the closures on R
   a. enumerate the attribute subsets in R2
   b. find the closure of the attribute subsets on R
   c. project these closures onto R2, by removing irrelevant attributes

$\{A\}^+=\{AB\}$	$\{C\}^+=\{C\}$
$\{D\}^+=\{D\}$	$\{E\}^+=\{E\}$
$\{AC\}^+=\{ABCD\}$	$\{AD\}^+=\{ABD\}$
$\{AE\}^+=\{ABE\}$	$\{CD\}^+=\{CD\}$
$\{CE\}^+=\{CE\}$	$\{DE\}^+=\{DE\}$
$\{ACD\}^+=\{ABCD\}$	$\{ACE\}^+=\{ABCDE\}$
$\{ADE\}^+=\{ABCDE\}$	$\{CDE\}^+=\{CDE\}$

{AC}+ violates the “more but not all” property → decompose further

- R(A, B, C, D) with FDs BC→D, D→A, A→B

1. Find a subset X of the attributes in R, such that its closure X^+ (i) contains more attributes than X, but (ii) does not contain all attributes in R

- $\{A\}^+ = \{A, B\}$

2. Decompose R into two tables R_1 and R_2, such that
   - R_1 contains all attributes in X^+
   - R_2 contains all attributes in X as well as the attributes not in X^+

- $R_1(A, B)$, $R_2(A, C, D)$

3. Check if R_1 and R_2 are in BCNF

- R_1: Yes. R_2: No
- Further decompose R2

- R(A, B, C, D) with FDs BC→D, D→A, A→B
- $R_1(A, B)$, $R_2(A, C, D)$
- Further decompose R_2

1. Find a subset X of the attributes in R_2, such that its closure X^+ (i) contains more attributes than X, but (ii) does not contain all attributes in R_2

- $\{A\}^+ = \{A\}$, $\{C\}^+ = \{C\}$, $\{D\}^+ = \{A, D\}$,

2. Decompose R_1 into two tables R_3 and R_4, such that
   - R_3 contains all attributes in X^+
   - R_4 contains all attributes in X as well as the attributes not in X^+

- $R_3(A, D)$, $R_4(C, D)$

3. Check if R_3 and R_4 are in BCNF

- Yes. Final results: $R_1(A, B)$, $R_3(A, D)$, $R_4(C, D)$

Lossless Join Decomposition

- a decomposition guarantees lossless join whenever the common attribute in R1 and R2 constitute a superkey of R1 or R2
- A decomposition is lossless join if there exists a sequence of binary lossless-join decompositions that generates that decomposition.
  - [Tut 9] Schema R(A, B, C, D, E) with FDs F = {AB → C, AC → D, E → ABCD} and decomposition {R1(A, B, C), R2(A, B, E), R3(A, C, D)}.
  - First, decompose R into R3(A,C,D) and R4(A,B,C,E). This is lossless because R3 ∩ R4 = AC, and AC is a superkey of R3 .
  - Next, decompose R4 into R1(A,B,C) and R2(A,B,E). This is also lossless because R1 ∩ R2 = AB, and AB is a superkey of R1.
  - Therefore, {R1(A,B,C), R2(A,B,E), R3 (A,C,D)} is a lossless-join decomposition.

L11: 3NF

- A table satisfies 3NF, if and only if for every non-trivial and decomposed FD
  - Either the left hand side is a superkey
  - Or the RHS is a prime attribute (i.e., it appears in a key)

Dependency Preservation

- decomposition preserves all FDs, iff S and S' are equivalent
  - Every FD in S' can be derived from S
  - Every FD in S can be derived from S'

- S = {A→C, AC→D, E→AD, E→H}
- S' = {A→CD, E→AH}
- Prove that S and S' are equivalent
- First, prove that S' can be derived from S
  - Given S, we have $\{A\}^+$ = {ACD}, so A→CD is implied by S
  - Given S, we have $\{E\}^+$ = {EADHC}, so E→AH is implied by S
  - Hence, S' can be derived from S

- S = {A→C, AC→D, E→AD, E→H}
- S' = {A→CD, E→AH}
- Prove that S and S' are equivalent
- Second, prove that S can be derived from S'
  - Given S', we have $\{A\}^+$ = {ACD}, so A→C is implied by S'
  - Given S', we have $\{AC\}^+$ = {ACD}, so AC→D is implied by S'
  - Given S', we have $\{E\}^+$ = {EADHC}, so E→AD and E→H are implied by S'
  - Hence, S can be derived from S'

- Schema R(A, B, C, D), with A→BCD, C→D
- Decomposition: R1(A, C), R2(A, B, D)
- Closures on R1:
  - $\{A\}^+$ = {ABCD}, $\{C\}^+$ = {CD}
- So we have A→C on R1
- Closures on R2:
  - $\{A\}^+$ = {ABCD}, $\{B\}^+$ = {B}, $\{D\}^+$ = {D}
- So we have A→BD on R2
- Given A→C and A→BD, we have
  - $\{A\}^+$ = {ABCD}, $\{C\}^+$ = {C}
- So C→D is not preserved by the decomposition

- Schema R(A, B, C, D, E), with AB→C, AC→D, E→ABCD
- Decomposition: R1(A, B, C), R2(A, B, E), R2(A, C, D)
- Closures on R1:
  - $\{A\}^+$ = {A}, $\{B\}^+$ = {B}, $\{C\}^+$ = {C}
  - $\{AB\}^+$ = {ABCD}, $\{AC\}^+$ = {ACD}, $\{BC\}^+$ = {BC}
- So we have AB→C on R1
- Closures on R2:
  - $\{A\}^+$ = {A}, $\{B\}^+$ = {B}, $\{E\}^+$ = {EABCD}
  - $\{AB\}^+$ = {ABCD}
- So we have E→AB on R2
- Closures on R3:
  - $\{A\}^+$ = {A}, $\{C\}^+$ = {C}, $\{D\}^+$ = {D}
  - $\{AC\}^+$ = {ACD}, $\{AD\}^+$ = {AD}, $\{CD\}^+$ = {CD}
- So we have AC→D on R3
- Given AB→C, E→AB, and AC→D, we have
  - $\{AB\}^+$ = {ABCD}, $\{AC\}^+$ = {ACD}, $\{E\}^+$ = {EABCD}
- So all FDs on R (i.e., AB→C, AC→D, E→ABCD) are preserved

3NF Check

1. Compute the closure for each subset of the attributes in R
2. Derive the keys of R
3. For each closure $\{X_1, ..., X_k\}^+ = \{Y_1, ..., Y_m\}$, check if
   - $\{Y_1, ..., Y_m\}$ does not contain all attributes, and
   - there is an attribute in $\{Y_1, ..., Y_m\}$ that is not in $\{X_1, ..., X_K\}$ and is not a prime attribute
4. If such a closure does not exist, then R is in 3NF

- R(A, B, C, D) with FDs B → C, B → D

  2. Derive the keys of R
     keys: AB
  - $\{A\}^+ = \{A\}$, $\{B\}^+ = \{BCD\}$, $\{C\}^+ = \{C\}$, $\{D\}^+ = \{D\}$
  - $\{AB\}^+ = \{ABCD\}$, $\{AC\}^+ = \{AC\}$, $\{AD\}^+ = \{AD\}$
  - $\{BC\}^+ = \{BCD\}$, $\{BD\}^+ = \{BCD\}$, $\{CD\}^+ = \{CD\}$
  - $\{ABC\}^+ = \{ABD\}^+ = \{ABCD\}$
  - $\{BCD\}^+ = \{BCD\}$, $\{ACD\}^+ = \{ACD\}$
  - $\{ABCD\}^+ = \{ABCD\}$

  3. For each closure $\{X_1, ..., X_k\}^+ = \{Y_1, ..., Y_m\}$, check if
  (i) $\{Y_1, ..., Y_m\}$ does not contain all attributes, and
  (ii) there is an attribute in $\{Y_1, ..., Y_m\}$ that is not in $\{X_1, ..., X_K\}$ and is not a prime attribute

Not in 3NF

Minimal Basis (aka Minimal Cover)

Conditions

1. Every FD in the minimal basis can be derived from S, and vice versa.
2. Every FD in the minimal basis is a non-trivial and decomposed FD.
3. No FD in the minimal basis is redundant.
   - if any FD is removed from M, then some FD in S cannot be derived from M
4. For each FD in the minimal basis, none of the attributes on the left hand side is redundant
   - if we remove an attribute from the LHS, then the resulting FD cannot be derived from the original set of FDs

Algorithm for Minimal Basis

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

- Example: S = {A→BD, AB→C, C→D, BC→D}
- Step 1: Transform the FDs, so that each right hand side contains only one attribute
- Result: S = {A→B, A→D, AB→C, C→D, BC→D}
- Reason:
  - Condition 2 for minimal basis:
    Each FD is a non-trivial and decomposed FD

- Step 2: Remove redundant attributes on the left hand side of each FD
- Both AB→C and BC→D have more than one attribute on the lhs
- Let's check AB→C first
- Is A redundant?
- If we remove A, then AB→C becomes B→C
- Whether this removal is OK depends on whether B→C is implied by S
  - If B→C is implied by S, then the removal of A is OK, since the removal does not add extraneous information into S
- Is B→C implied by S?
- Check: Given S, we have $\{B\}^+ = \{B\}$, which does NOT contain C
- Therefore, B→C is not implied by S, and hence, A is NOT redundant

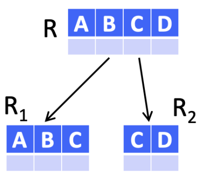
- Step 2: Remove redundant attributes on the left hand side of each FD
- Both AB→C and BC→D have more than one attribute on the lhs
- Let's check AB→C first
- Is B redundant?
- If we remove B, then AB→C becomes A→C
- Whether this is OK depends on whether A→C is implied by S
- Is A→C implied by S?
- Check: Given S, we have $\{A\}^+ = \{ABCD\}$, which contains C
- Therefore, A→C is implied by S, and hence, B is redundant in AB→C
- Thus, we can simplify AB→C to A→C
- Result: S = {A→B, A→D, A→C, C→D, BC→D}

- Step 3: Remove redundant FDs
- Is A→B redundant?
- i.e., is A→B implied by other FDs in S?
- Let's check
- Without A→B, we have {A→D, A→C, C→D}
- Given those FDs, we have $\{A\}^+ = \{ACD\}$, which does not contain B
- Therefore, A→B is not implied by the other FDs

- Step 3: Remove redundant FDs
- Is A→D redundant?
- i.e., is A→D implied by other FDs in S?
- Let's check
- Without A→D, we have {A→B, A→C, C→D}
- Given those FDs, we have $\{A\}^+ = \{ABCD\}$, which contains D
- Therefore, A→D is implied by the other FDs
- Hence, A→D is redundant and should be removed
- Result: S = {A→B, A→C, C→D}

3NF Decomposition Algorithm

- Given: A table R, and a set S of FDs
  - e.g., R(A, B, C, D)
    S = {A→BD, AB→C, C→D, BC→D}
- Step 1: Derive a minimal basis of S
  - e.g., a minimal basis of S is {A→B, A→C, C→D}
- Step 2: In the minimal basis, combine the FDs whose left hand sides are the same
  - e.g., after combining A→B and A→C, we have {A→BC, C→D}
- Step 3: Create a table for each FD remained
  - R_1(A, B, C), R_2(C, D)
- Step 4: If none of the tables contains a key of the original table R, create a table that contains a key of R (any key would do)



step 5: remove redundant tables, if any

R(A, B, C, D, E), with A→B, A→C, B→C, E→C, E→D

1. Minimal basis: A→B, B→C, E→C, E→D
- Combine the FDs whose left hand sides are the same: A→B, B→C, E→CD
- For each FD, construct a table that contains all attributes in the FD:
  R_1(A, B), R_2(B, C), R_3(C, D, E)
- Check if any of the tables contain a key for R; if not, then create a table that contains a key for R:
  Key for R is {AE}, which is not contained in R_1, R_2, or R_3.
- Create another table R_4(A, E)
- Final result: R_1(A, B), R_2(B, C), R_3(C, D, E), R_4(A, E)

- Given: A table R, and a set S of FDs
  - e.g., R(A, B, C, D), S = {AB→C, C→A, B→D}
- Step 1: Derive a minimal basis of S
  - The minimal basis of S is {AB→C, C→A, B→D}
- Step 2: In the minimal basis, combine the FDs whose left hand sides are the same
  - Nothing to be combined
- Step 3: Create a table for each FD remained
  - R_1(A, B, C), R_2(C, A), R3(B, D)
- Step 4: If none of the tables contains a key of the original table R, create a table that contains a key of R (any key would do)
  - R_1(A, B, C) already contains AB, one of the keys of R
- Almost done… But do we need R_2? Everything in R_2 is already in R_1
- Answer: No
- Final decomposition: R_1(A, B, C), R_3(B, D)

	BCNF	3NF
defintion	if every non-trivial and decomposed FD’s LHS is a superkey.	iff for every non-trivial and decomposed FD - Either the left hand side is a superkey - Or the RHS is a prime attribute (i.e., it appears in a key)
check	we have a violation of BCNF, iff we have a closure that satisfies the "more but not all" condition	iff we have a closure that satisfies the “more but not all” property and the extra attribute is not a prime attribute
decomposition	- 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	- only 1 split which divides the table into 2 or more parts
good properties	- no update/deletion/insertion anomalies - small redundancy - the original table can always be reconstructed from the decomposed tables (lossless join)	
bad properties	- FD may not be preserved	
	go for BCNF if we can find one BCNF decomposition that preservers all FDs, or if preserving all FDs are not important; else go for 3NF	more lenient than BCNF - satisfy BNCF → satisfy 3NF but not necessarily vice versa - violate 3NF → violate BCNF, but not necessarily vice versa