Universitetet i Sørøst-Norge

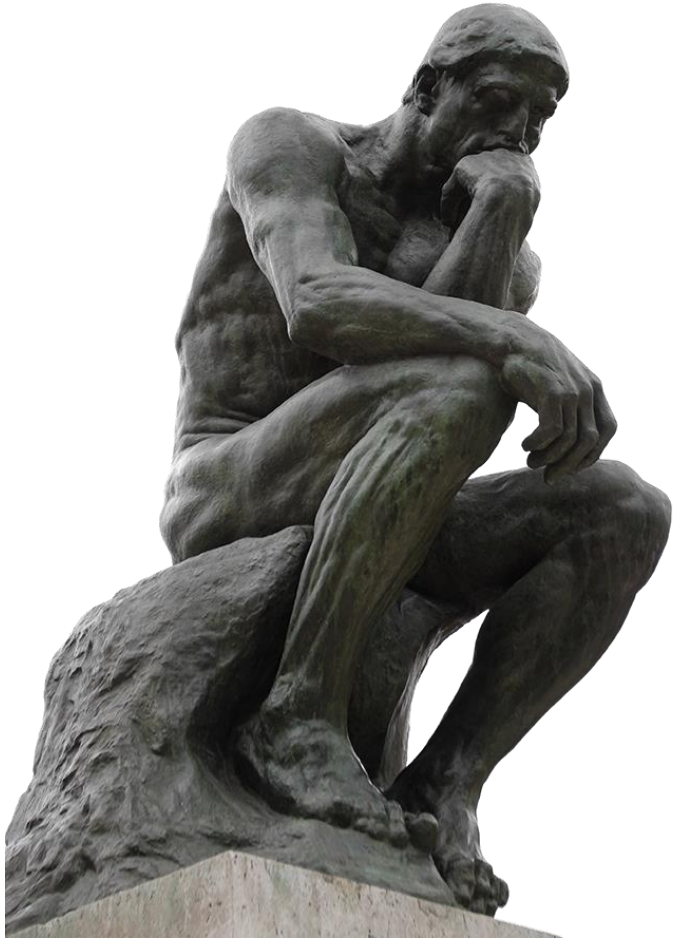
Leksjon 6: Lagrede programmer

Jarle Håvik

DAT2000 – Database 2

Kunnskapsmål

- Ha kjennskap til og kunne forklare begrepene:
  - lagret prosedyre,
  - lagret funksjon,
  - prosedyredeklarasjon
- kunne beskrive bruksområdet for lagrede rutiner
- kunne beskrive hvordan lagrede rutiner blir utført
- kunne forklare hvordan sikkerhet og rettigheter håndteres i lagrede rutiner



Ferdigheter

- kunne programmere enkle lagrede prosedyrer og funksjoner i et DBMS (PostgreSQL/MySQL)
- kunne bruke (kalle) lagrede prosedyrer og funksjoner fra SQL
- Kunne bruke lagrede prosedyrer og funksjoner fra PHP (kommer i APP2000).
- kunne hente ut informasjon om lagrede rutiner fra systemkatalogen i DBMS

Lagrede rutiner i DBMS
Ser på både PostgreSQL og MySQL

- Definisjon av lagrede rutiner
  - Lagrede prosedyrer
  - Lagrede funksjoner
  - Parameteroverføring: IN, OUT, INOUT,
  - Returverdier
- SQL-spørringer i lagrede rutiner
  - INSERT, UPDATE, DELETE
  - SELECT INTO
  - CURSOR
- Klargjorte spørringer (prepared statements)

Litteratur

- Silberschatz, Abraham et.al: Database System Concepts
  - Kapittel 5 Advanced SQL s, 183 – 205 (hovedvekt på kapittel 5.2)
- Kristoffersen, Bjørn: Databasesystemer
  - Kapittel 13 Lagrede programmer (s. 375 – 390)
- Dubois, Paul: MySQL
  - kapittel 4.2.1. Compound Statements and Statement Delimiters
  - kapittel 4.2.2. Stored Functions and Procedure
  - kapittel 4.3. SECURITY FOR VIEWS AND STORED PROGRAMS

Nettressurser

- An Introduction to PostgreSQL PL/pgSQL Procedural Language
  https://www.postgresqltutorial.com/introduction-to-postgresql-stored-procedures/
- Stored Procedures and Functions in PostgreSQL - Getting Started
  - http://www.sqlines.com/postgresql/stored_procedures_functions
- Introduction to PostgreSQL PL/pgSQL
  - https://www.postgresqltutorial.com/introduction-to-postgresql-stored-procedures/
- PostgreSQL: Documentation 11: CREATE PROCEDURE
  - https://www.postgresql.org/docs/11/sql-createprocedure.html
- Stored Procedures in PG 11 – Better late then never
  - https://www.highgo.ca/2020/04/10/stored-procedures-in-pg-11-better-late-then-never/

Videoer

- Linkedin Learning:
- Prepared-statements-and-stored-procedures (1:43)

Hva er SQL

- SQL er et standardisert relasjonelt databasespråk som lar brukerne opprette, spørre og kontrollere databaseobjekter.
  - SQL er et deklarativt språk
  - Høg-nivå spark (3GL) som C er *prosedurale spark.*

- Step 1:
  - Find ID and text of all page written by Bob and containing the word «db»

Step2:

```
SELECT p.id, p.text
FROM posts AS p, users AS u
WHERE u.id = p.authorId
      AND u.name='Bob'
      AND p.text ILIKE '%db%';
```

users

id	name	karma
729	Bob	35
730	John	0

posts

id	text	ts	authorId
33981	'Hello DB!'	1493897351	729
33982	'Show me code'	1493904323	812

Hvordan blir en spørring besvart?

```
SELECT p.id, p.text
FROM posts AS p, users AS u
WHERE u.id = p.authorId
      AND u.name='Bob'
      AND p.text ILIKE '%db%';
```

(p, u)

p.id	p.text	p.ts	p.authorId	u.id	u.name	u.karma
33981	'Hello DB!'	...	729	729	Bob	35
33981	'Hello DB!'	...	729	730	John	0
33982	'Show me code'	...	812	729	Bob	35
33982	'Show me code'	...	812	730	John	0

p

id	text	ts	authorId
33981	'Hello DB!'	...	729
33982	'Show me code'	...	812

u

id	name	karma
729	Bob	35
730	John	0

Hvordan blir en spørring besvart?

```
SELECT p.id, p.text
FROM posts AS p, users AS u
WHERE u.id = p.authorId
      AND u.name='Bob'
      AND p.text ILIKE '%db%';
```

where(p, u)

p.id	p.text	p.ts	p.authorId	u.id	u.name	u.karma
33981	'Hello DB!'	...	729	729	Bob	35

(p, u)

p.id	p.text	p.ts	p.authorId	u.id	u.name	u.karma
33981	'Hello DB!'	...	729	729	Bob	35
33981	'Hello DB!'	...	729	730	John	0
33982	'Show me code'	...	812	729	Bob	35
33982	'Show me code'	...	812	730	John	0

Hvordan blir en spørring besvart?

```
SELECT p.id, p.text
FROM posts AS p, users AS u
WHERE u.id = p.authorId
      AND u.name='Bob'
      AND p.text ILIKE '%db%';
```

select(where(p, u))

p.id	p.text
33981	'Hello DB!'

where(p, u)

p.id	p.text	p.ts	p.authorId	u.id	u.name	u.karma
33981	'Hello DB!'	...	729	729	Bob	35

Structured Query Language (SQL)

- Data Definition Language (DDL) på skjema
  - CREATE TABLE …
  - ALTER TABLE …
  - DROP TABLE …
- Data Manipulation Language (DML) på tabeller
  - INSERT INTO … VALUES …
  - SELECT … FROM … WHERE …
  - UPDATE … SET … WHERE …
  - DELETE FROM … WHERE …

Structured Query Language (SQL)

- Transaction Control Language (TCL): brukes til å håndtere endringer som påvirker dataene
  - COMMIT…
  - ROLLBACK….
  - SAVEPOINT…
- Data Control Language (DCL): brukes til å gi sikkerhet til databaseobjekter.
  - GRANT…
  - REVOKE…

SQL mangler programkonstruksjoner, men dette har lagrede prosedyrer!
Programkonstruksjoner?

Programmerings-konstruksjoner

Kontrollstrukturer

- Betinget IF-setning
- Betinget CASE-setning
- Iterasjonsstrukturer
  - LOOP
  - WHILE og REPEAT
  - FOR

```
IF (position = 'Manager') THEN
        salary := salary*1.05;
ELSE
        salary := salary*1.02;
END IF;
```

```
UPDATE Staff
SET salary = CASE
                WHEN position = 'Manager'
                THEN salary * 1.05
                ELSE
                salary * 1.02
END;
```

```
WHILE (condition) DO
                <SQL statement list>
END WHILE [labelName];
REPEAT
                <SQL statement list>
UNTIL (condition)
END REPEAT [labelName];
```

```
myLoop1:
FOR iStaff AS SELECT COUNT(*) FROM
PropertyForRent WHERE staffNo = 'SG14'
DO
                .....
END FOR myLoop1;
```

```
x:=1;
myLoop:
LOOP
                x := x+1;
                IF (x > 3) THEN
                        EXIT myLoop; --- exit
                        loop now
END LOOP myLoop;
--- control resumes here
y := 2;
```

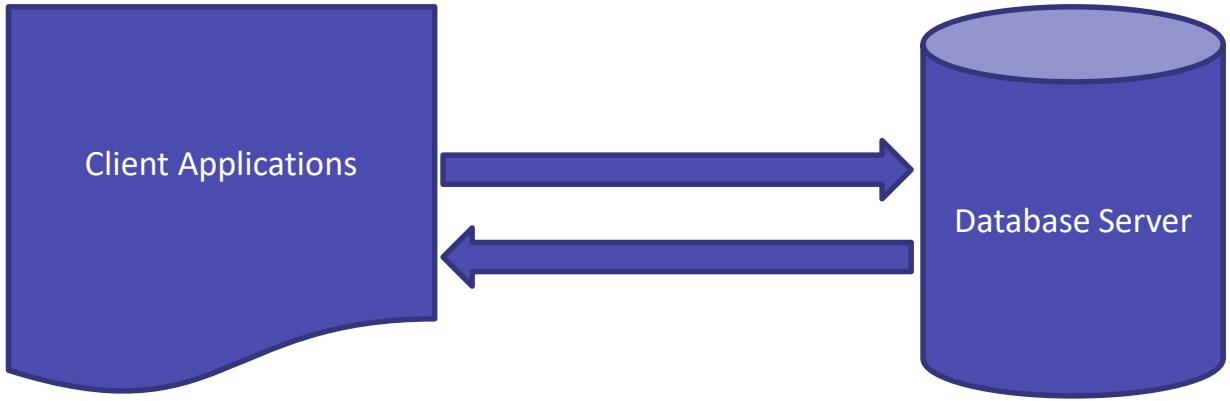
Inline SQL vs. Stored procedure:

Enten er en inline SQL-setning innebygd i applikasjonskoden.

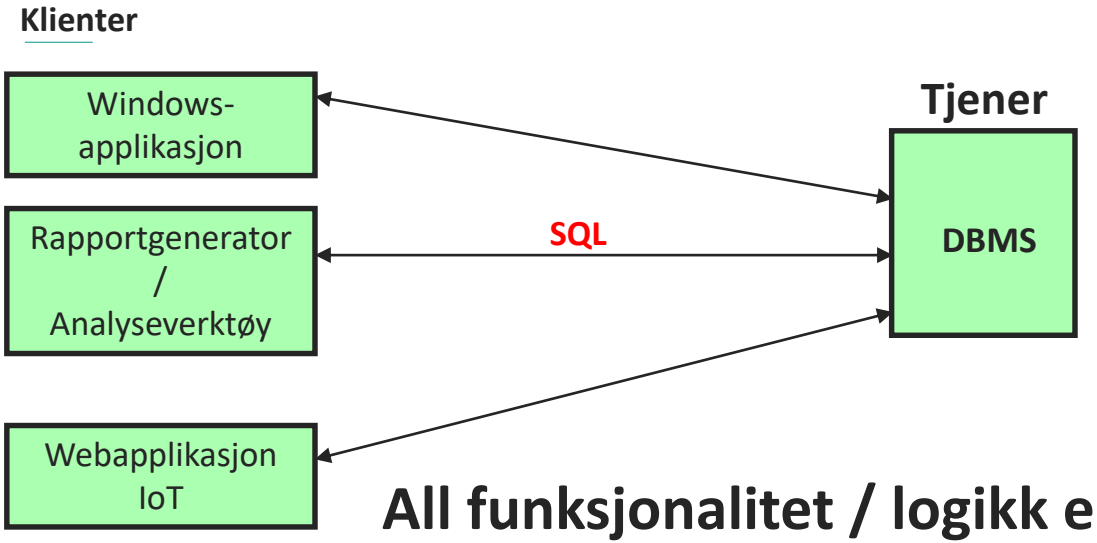
Deretter må du sende SQL-kommandoene til serveren og behandle resultatene.

ELLER en lagret rutine som er lagret på serveren.

Deretter kan du opprette applikasjoner på klientsiden som utfører den lagrede rutinen og behandler resultatene.



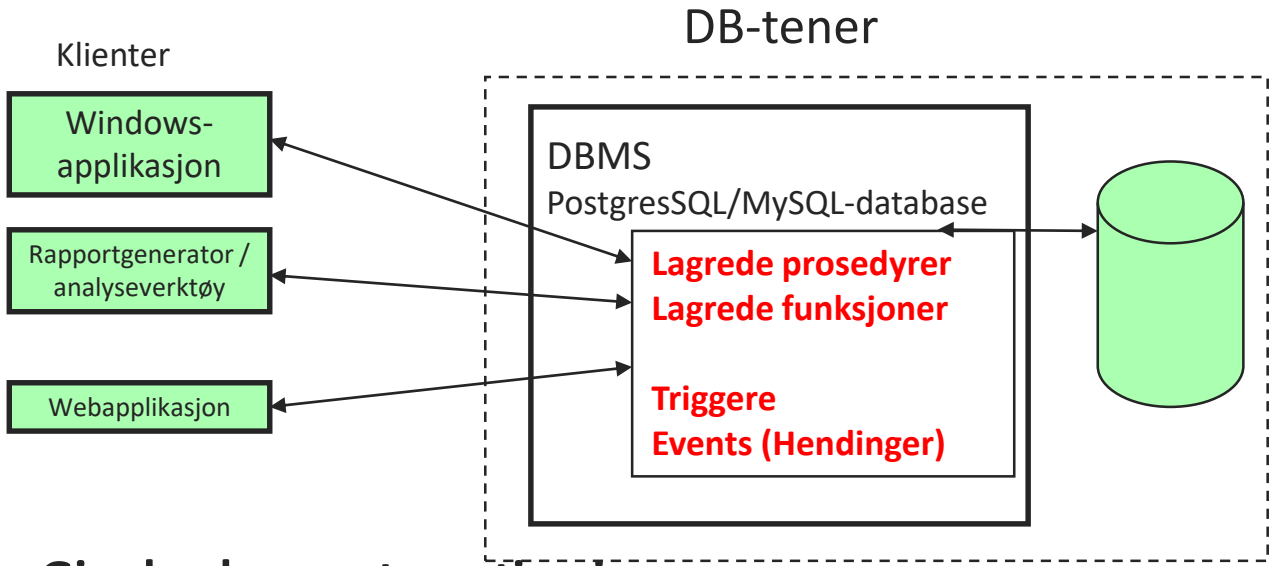
«Tradisjonell» klient / tjener - arkitektur



All funksjonalitet / logikk er programmert i klienten

- DBMS brukes kun til lagring / henting av data med SQL
- Samme metode (algoritme) må ofte lages i flere klientprogrammer
- Endringer i databasestruktur krever endringer i flere klientprogrammer

Databaseprogrammer i DBMS



Gjenbruk og automatisering

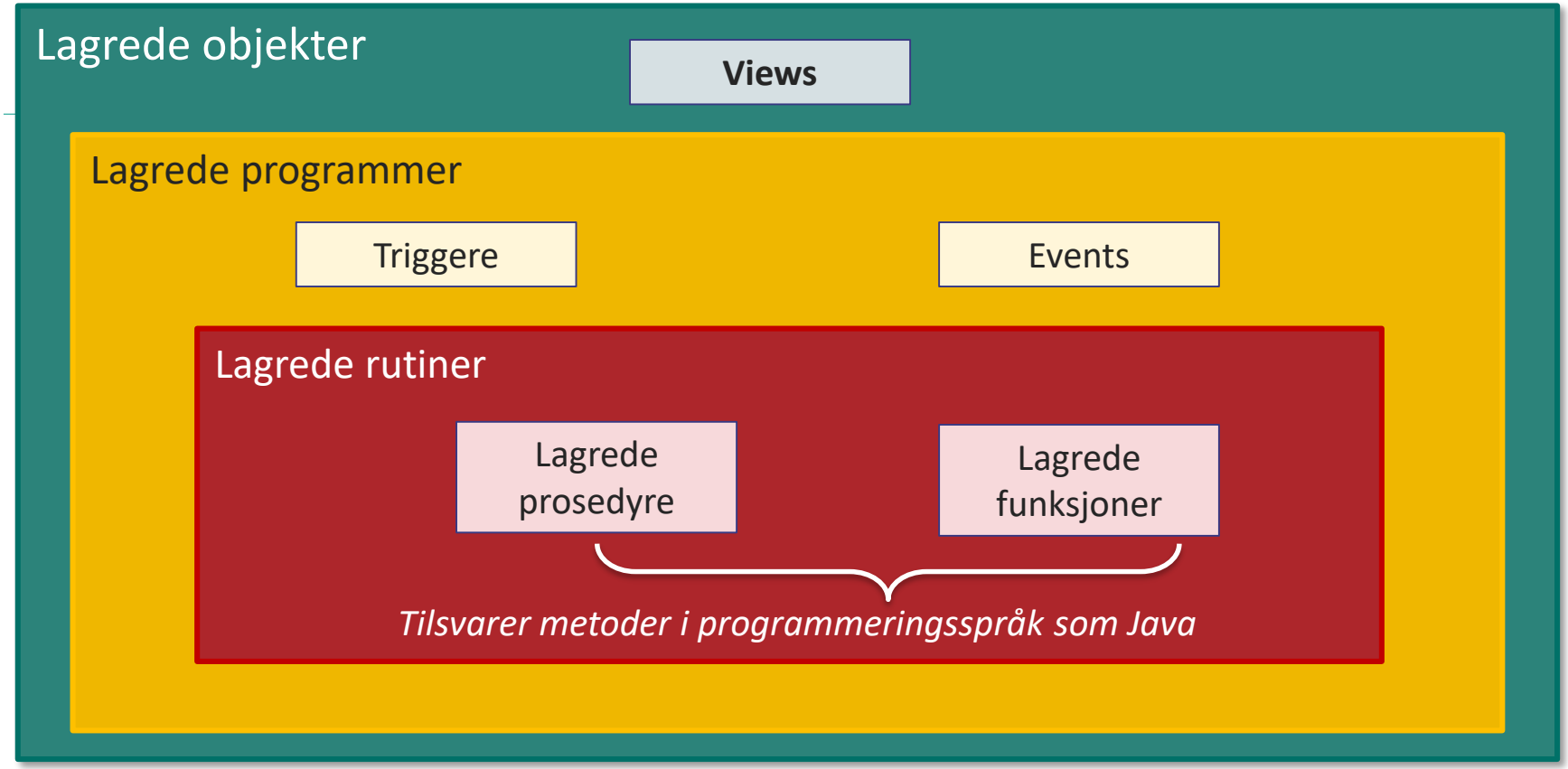
– Lagrede prosedyrer kjører på tjeneren og kan kalles fra flere klienter

– Kan redusere nettverkstrafikk

– Triggere aktiveres automatisk ved endringer i databasen

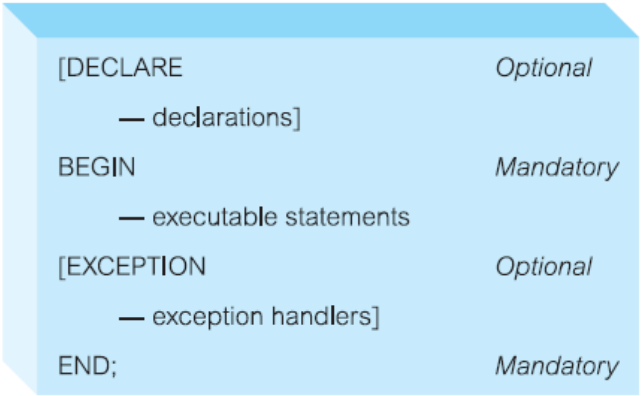
– Kan kombineres med bruk av SQL fra klienten

Lagrede objekter i DBMS



Hva er PL/SQL?

- PL/SQL står for Procedural Language utvidelse av SQL.
- PL / SQL er en kombinasjon av SQL sammen med prosessuelle funksjoner i programmeringsspråk
- Utviklet av Oracle Corporation tidlig på 90-tallet for å forbedre kapasiteten til SQL.
- PL / SQL er et blokkstrukturert språk, blokker kan være helt separate eller nøstet i hverandre.



Prosessuelle og Ikke-prosessuelle språk

- Prosessuelt språk:

  Et språk som lar brukeren fortelle systemet ***hvilke*** data som trengs og nøyaktig ***hvordan*** de skal hente dataene.

- Ikke-prosessuelt språk:

  Et språk som lar brukeren oppgi ***hvilke*** data som trengs i stedet for hvordan de skal hentes.

https://no.wikipedia.org/wiki/Programmeringsparadigme

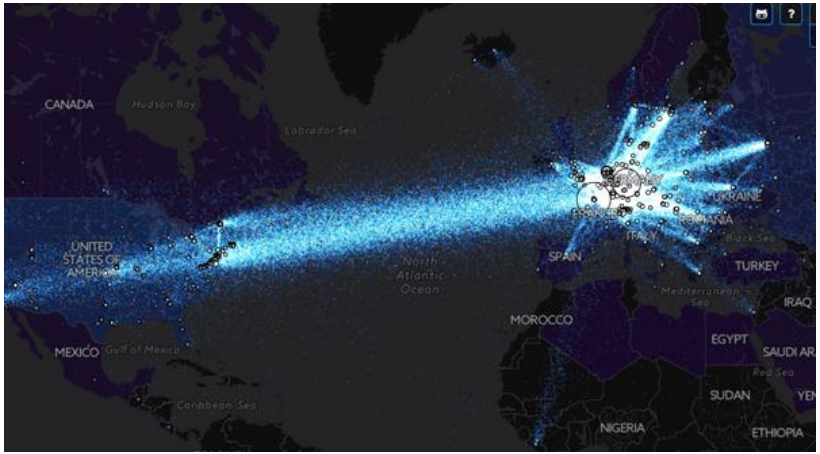
Fordeler med lagrede rutiner

1. Overhead

Lagrede prosedyresetninger lagres direkte i databasen, de kan redusere kompileringsomkostningene som vanligvis er nødvendig når programvareapplikasjoner sender innebygde SQL-spørsmål til en database.

Lagrede prosedyrer kompileres bare en gang i løpet av opprettelsen, og de lagres i serveren. Derfor er gjentatt kompilering ikke nødvendig under gjennomføring av prosedyren, og øker dermed hastigheten på utførelsen. Imidlertid, hvis det gjøres oppdateringer i prosedyren, blir den kompilert på nytt.

Fordeler med lagrede rutiner



2. Redusert nettverkstrafikk

Lagrede rutiner kan kjøres direkte i databasemotoren, dette betyr at rutiner kjøres helt på en spesialisert databaseserver, som har direkte tilgang til dataene du får tilgang til.

Fordelen her er at nettverkskommunikasjonskostnader kan unngås helt. Dette blir viktigere for komplekse serier av SQL-setninger.

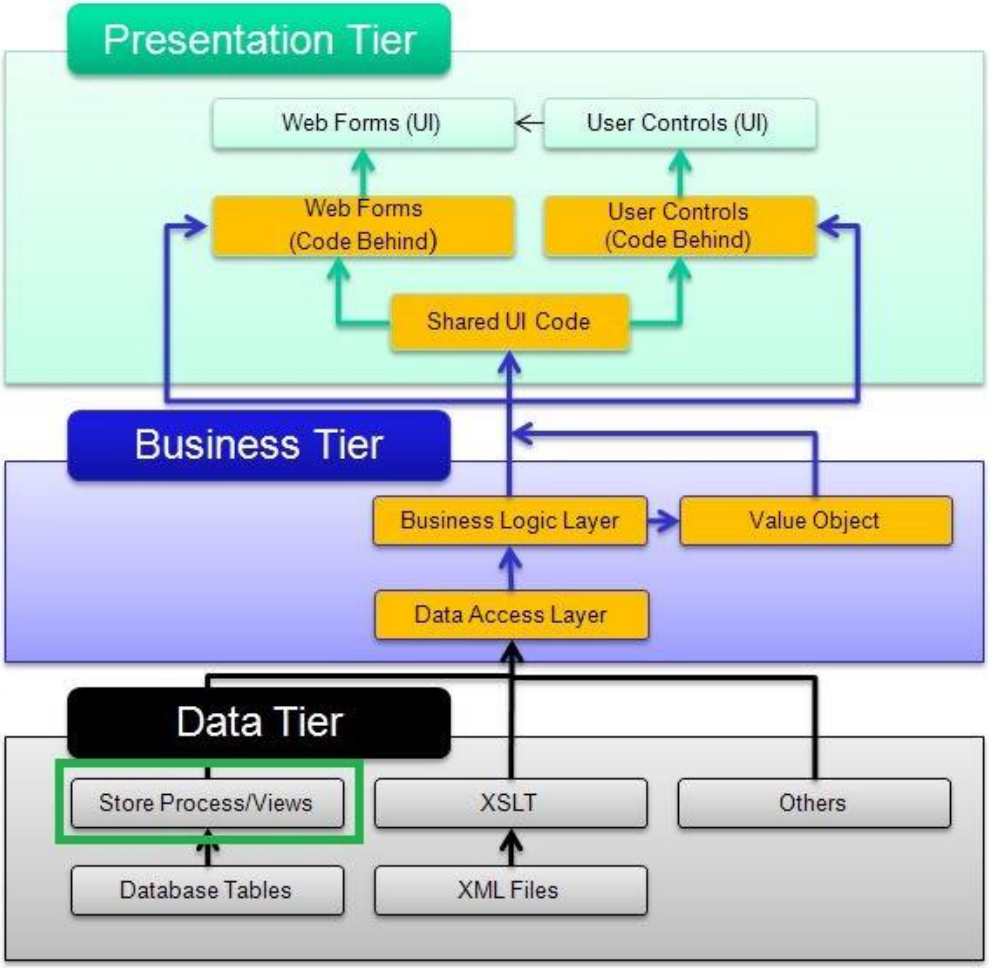
Fordeler med lagrede rutiner

3. Innkapsling av forretningslogikk

Lagrede prosedyrer tillater programmerere å legge inn forretningslogikk som et API i databasen, noe som kan forenkle datahåndteringen og redusere behovet for å kode logikken andre steder i klientprogrammer.

Dette kan redusere datakorrupsjon ved feil på klientprogrammer.
Databasesystemet kan sikre dataintegritet og konsistens ved hjelp av lagrede prosedyrer.

Business Logic



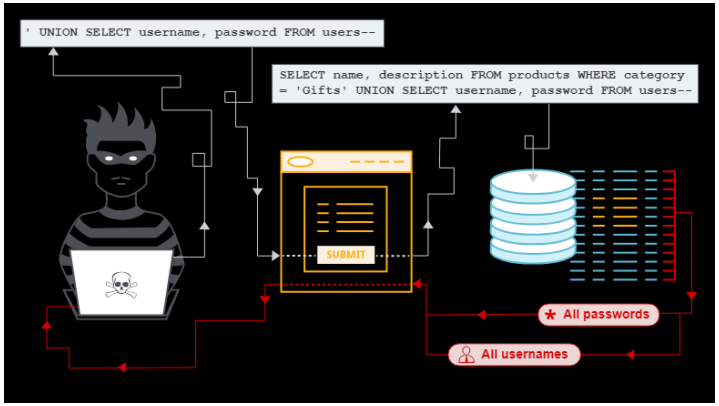
Fordeler med lagrede rutiner

4. Sikkerhet

Lagrede rutiner kan opne for DML til de brukerne som har tillatelse til å utføre dem, noe som gir mer kontroll.

Selv om en konto blir kompromittert, kan angriperen fremdeles ikke utføre vilkårlige kommandoer mot dataene, og kan fremdeles bare få tilgang til en begrenset delmengde av data.

Fordeler med lagrede rutiner



Lagrede rutiner er motstandsdyktige mot SQL-injeksjonsangrep (som er vanlige angrep på applikasjonsnivå), der en bruker kan entre skadelig kode i et av feltene til et webprogram for å utføre vilkårlige kommandoer mot databasen.

SQL-injections er en kombinasjon av SQL-setninger og teknikker som kan bryte inn i databaser og gjøre uautorisert manipulering av dataene.

Lagrede rutiner

- Lagret prosedyre
  - Registrere ny ansatt
  - Data om den ansatte er parametre
  - Kontrollerer inndata og setter inn ny rad hvis alt er ok
  - Kan sende tilbake eventuell feilmelding i en ut-parameter
- Lagret funksjon
  - Gjennomsnittspris for varer i en kategori
  - Kategorinummer er parameter
  - Prisen blir returverdi
- Lagret rutine
  - Funksjon eller prosedyre

3 typer parameteroverføring

- IN - parametre
  - Er standard og svarer til verdioverføring i Java/C
  - Benyttes for å sende verdier inn til prosedyren
- OUT - parametre
  - Benyttes for å returnere verdier fra prosedyren
  - Hensiktsmessig hvis man vil returnere mer enn én verdi
- IN OUT er en kombinasjon av IN og OUT
  - Variabelverdien sendes inn i prosedyren og kan endres der
  - Ev. endret verdi sendes tilbake fra prosedyren
- Merk:
  - Variable som IN-parametre endrer ikke verdi som følge av prosedyrekallet
  - Variable som OUT og IN OUT parametre kan endre verdi

Definisjon og kall av lagret prosedyre

- Først et litt atypisk eksempel som ikke bruker databasen, men som illustrerer inn- og ut-parametre godt.

```
CREATE PROCEDURE intdiv
(
    IN  x INT,
    IN  y INT,
    OUT d INT,
    OUT r INT
)
BEGIN
    SET d = x DIV y;
    SET r = x MOD y;
END
```

Prosedyrekall
(fra en annen prosedyre)

```
DECLARE svar INT;
DECLARE rest INT;
CALL intdiv(7,3,svar,rest);
-- svar er 2, rest er 1
```

Test av parameteroverføring

Parametertype angis i listen av formelle parametre

```
CREATE PROCEDURE param_test
(
  p_1 IN INT,
    p_2 OUT INT,
    p_3 IN OUT TEXT
)
BEGIN
    SET p_2 = p_1 + 1;
    SET p_3 = CONCAT(p_3, '...mer tekst.');
END;
```

- Forsøk på å tilordne p_1 en verdi gir feilmelding

Redefinere skilletegn

Semikolon i en lagret prosedyre skal ikke tolkes som at CREATE PROCEDURE er avsluttet…

```
1. mysql> DELIMITER $$
2. mysql> DROP PROCEDURE IF EXISTS intdiv $$
3. mysql> CREATE PROCEDURE intdiv … END $$
4. mysql> DELIMITER ;
```

Brukervariabler

«Sesjonsvariabler» kan være nyttige under testing (her vist med MySQL).

```
SET @x = 7;
SET @y = 3;

CALL intdiv(@x, @y, @d, @r);

SELECT @d divisjon, @r rest;
```

Resultat:

```
divisjon   rest
---------------
2           1
```

Bruk av SELECT for test-utskrift

```
DELIMITER $$
DROP PROCEDURE IF EXISTS skriv2 $$
CREATE PROCEDURE skriv2()
BEGIN
  DECLARE x INT;
  SET x = 2;
  SELECT CONCAT('Tallet er: ', x);
END $$
DELIMITER ;

CALL skriv2();
Tallet er: 2
```

SELECT uten FROM er lov:

```
SELECT 2+2;
```

Nyttig for å teste ut funksjoner.

Lagret funksjon

```
DROP FUNCTION IF EXISTS minst $$

CREATE FUNCTION minst(x INT, y INT)
  RETURNS INT
  DETERMINISTIC
BEGIN
  IF x<y THEN
    RETURN x;
  ELSE
    RETURN y;
  END IF;
END $$
```

Funksjonskall for å teste:

```
SELECT minst(5,2);
```

Alternativer til DETERMINISTIC:

```
NOT DETERMINISTIC
CONTAINS SQL
NO SQL
READS SQL DATA
MODIFIES SQL DATA
```

SQL i lagrede rutiner

- 3 typer:
  - Utvalgsspørringer som gir 1 rad (SELECT INTO)
  - Utvalgsspørrringer som kan gi flere rader – cursors
  - Øvrige spørringer (INSERT, UPDATE, DELETE,…) enklest – skriv SQL rett inn i koden:

```
CREATE PROCEDURE store_bokstaver()
BEGIN
  UPDATE kunde
  SET fornavn=UPPER(fornavn),
      etternavn=UPPER(etternavn);
END $$
```

SELECT INTO

Hvis en SQL-spørring alltid gir 1 rad kan verdiene kopieres over i lokale variabler vha SELECT INTO:

```
DECLARE v_pris NUMBER(8, 2);

SELECT pris_pr_enhet INTO v_pris
FROM   vare
WHERE  vnr = '10830';
```

Vet at vi får 1 rad?

- Likhet
- vnr er primærnøkkel

Cursors og spørreresultater

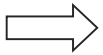
Cursor

- En peker som refererer en av radene i spørreresultatet.

For å behandle en databasetabell eller et spørreresultat:

- Ei løkke
- En «rad-peker» som flyttes
- Lese verdier i én og én rad

```
SELECT snr, navn,
  adresse,fdato
FROM student;
```



cursor

snr	navn	adresse	fdato
1	EVA	AVEIEN1	01.02.97
2	OLA	BVEIEN 2	10.02.03
5	KARI	AVEIEN 4	02.10.99
7	PER		02.10.92
9	GURI	CVEIEN 5	15.11.97

Bruk av cursors

Deklarerer cursoren

```
CURSOR c1 IS SELECT snr, navn FROM student;
```

Åpner cursoren

```
OPEN c1;
```

Henter én og én rad i en løkke til det ikke er flere rader

```
LOOP
  FETCH c1 INTO v_snr, v_navn;
  -- Hopp ut av løkka hvis forbi siste rad …
END LOOP;
```

Lukker cursoren

```
CLOSE c1;
```

Cursors del 1

```
CREATE PROCEDURE vis_varer()
BEGIN
  DECLARE ferdig INT DEFAULT FALSE;

  DECLARE v_vnr   CHAR(5);
  DECLARE v_betegnelse VARCHAR(20);

  DECLARE v_utdata TEXT DEFAULT ' ';

  DECLARE c_vareliste CURSOR FOR
    SELECT vnr, betegnelse FROM Vare;

  DECLARE CONTINUE HANDLER
    FOR NOT FOUND SET ferdig = TRUE;
```

Cursors del 2

```
    OPEN c_vareliste;
    les_varer: LOOP
       FETCH c_vareliste INTO v_vnr, v_betegnelse;
       IF ferdig THEN
         LEAVE les_varer;
       END IF;
       SET v_utdata = CONCAT(v_utdata, ' ', v_vnr,
                      ' ', v_betegnelse, '\n');
    END LOOP;
    CLOSE c_vareliste;

    SELECT v_utdata;
END $$
```