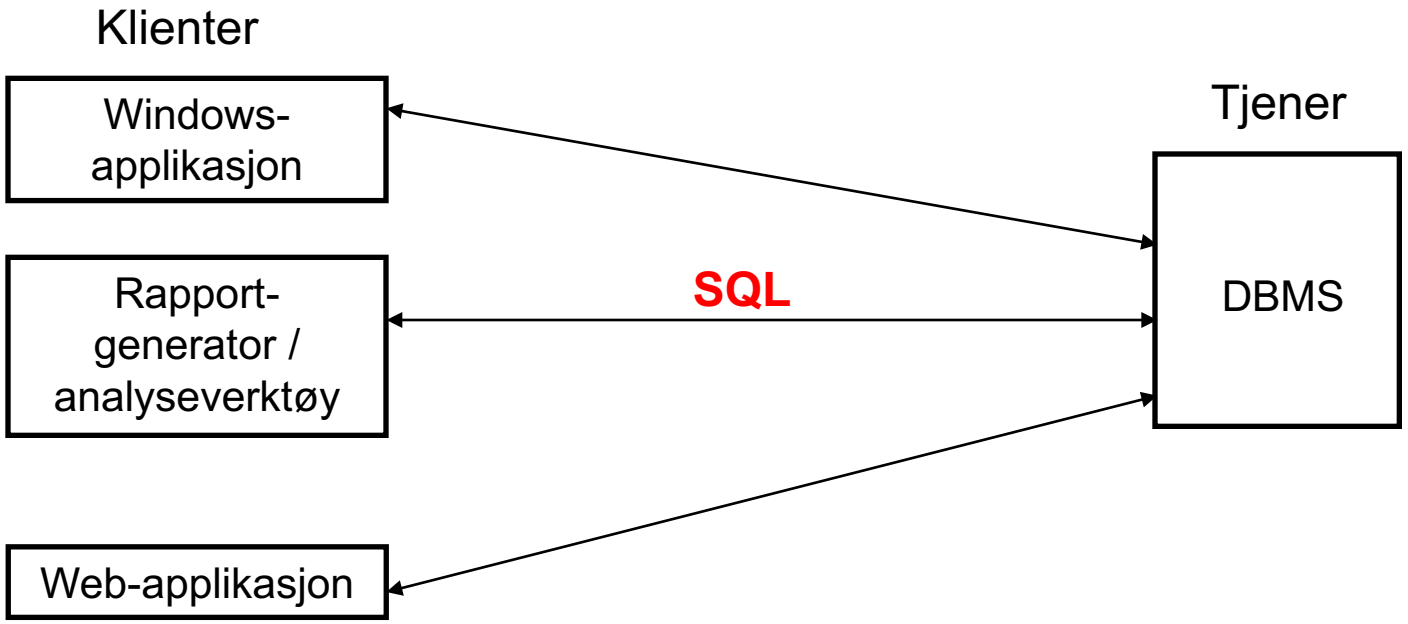
13

Lagrede programmer

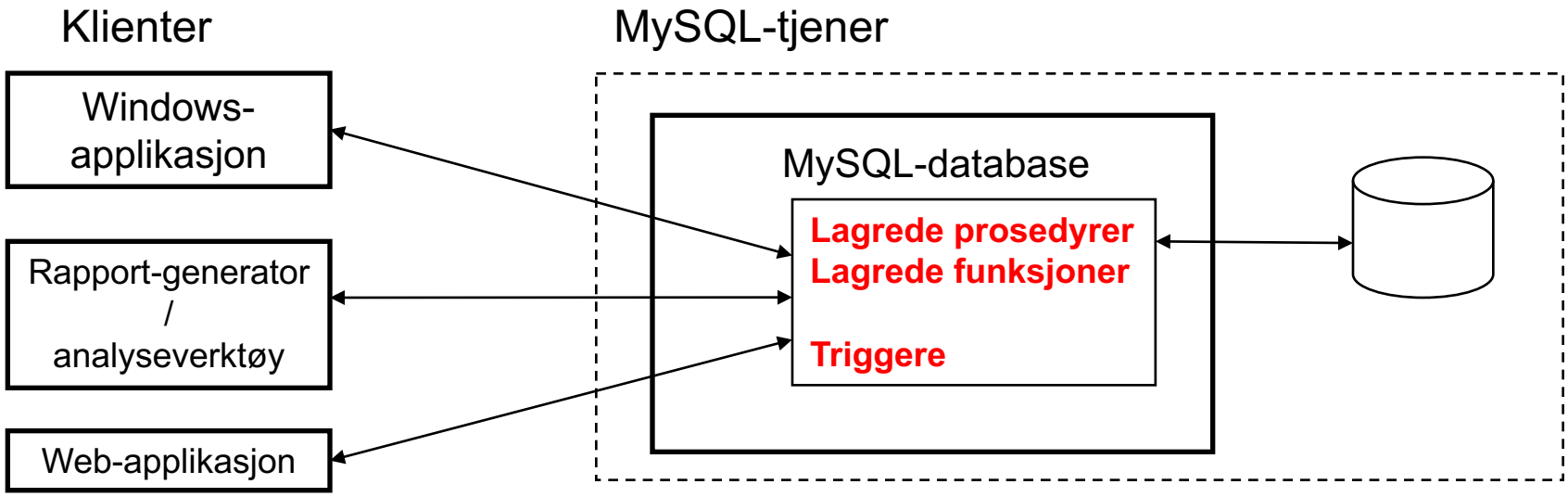
«Tradisjonell» klient / tjener - arkitektur



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 MySQL



Gjenbruk og automatisering

– Lagrede prosedyrer kjører på tjeneren og kan kalles fra alle klienter
– Kan redusere nettverkstrafikk
– Triggere aktiveres automatisk ved endringer i databasen
– Kan kombineres med bruk av SQL fra klienten

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
- Evt. endret verdi sendes tilbake fra prosedyren

Merk:

- Variable som IN-parametre endrer ikke verdi som følge av prosedyrekallet
- Variable som OUT og IN OUT parametere kan endre verdi

Definisjon og kall av lagret prosedyre

Prosedyredefinisjon

```
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…

```
mysql> DELIMITER $$

mysql> DROP PROCEDURE IF EXISTS intdiv $$

mysql> CREATE PROCEDURE intdiv … END $$

mysql> DELIMITER ;
```

Brukervariabler

En slags MySQL «sesjonsvariabler» nyttige under testing.

```
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 – markører (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

Markører og spørreresultater

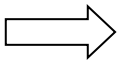
Markør (cursor)

- 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;
```



markør

snr	navn	adresse	fdato
1	EVA	AVEIEN1	1.2.70
2	OLA	BVEIEN 2	10.2.72
5	KARI	AVEIEN 4	2.10.65
7	PER		2.10.69
9	GURI	CVEIEN 5	15.11.60

Bruk av markører

Deklarerer markøren

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

Åpner markøren

```
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 markøren

```
CLOSE c1;
```

Triggere

Databasehendelser

Avfyring av triggere

Rad-triggere

(Setnings-triggere)

BEFORE-triggere

AFTER-triggere

OLD og NEW

Bruksområder for triggere

Triggere og PHP

A **database trigger** is procedural code that is **automatically executed** in response to certain **events** on a particular **table** […] in a database. The trigger is mostly used for keeping the **integrity** of the information on the database. [Wikipedia]

Hva er en trigger?

Et (lite) program som blir utført i databasen

- Kjøres automatisk som følge av en hendelse i databasen
- Mulige hendelser: INSERT, UPDATE, DELETE
- Vi sier at triggeren **avfyres**

Hvordan lage en trigger?

```
CREATE TRIGGER triggernavn
   [BEFORE | AFTER]
   [INSERT | UPDATE | DELETE] ON tabellnavn
   [FOR EACH ROW]
BEGIN
  setninger
END
```

Rad-trigger

Blir avfyrt for hver rad som blir berørt av en oppdatering.

```
CREATE TRIGGER tr_ansatt_1
   BEFORE UPDATE ON ansatt
   FOR EACH ROW
BEGIN
   -- Behandle raden
END
```

Husk at SQL-setninger kan oppdatere mange rader.

```
UPDATE ansatt SET timeloenn = timeloenn * 1.1
  WHERE avdnr = 2;
```

(Setnings-triggere)

Blir avfyrt én gang for hver SQL-setning

- Setnings-triggere er ikke støttet i MySQL

```
CREATE TRIGGER tr_ansatt_2
  BEFORE UPDATE ON ansatt
BEGIN
  -- Setninger
END
```

Kan for eksempel brukes til aksesskontroll

Automatisk formatering – OLD og NEW

```
CREATE TRIGGER tr_ansatt
   BEFORE UPDATE ON Ansatt
   FOR EACH ROW
BEGIN
   SET NEW.Fornavn = UPPER(NEW.Fornavn);
   SET NEW.EndretDato = CURDATE();
END
```

Kjører følgende oppdatering 27. februar 2016:

```
UPDATE Ansatt SET Fornavn = 'Petter'
WHERE AnsNr = 2;
```

OLD

AnsNr	2
Fornavn	Per
Timeloenn	250,00
AvdNr	1
EndretDato	12.1.2015

NEW

AnsNr	2
Fornavn	Petter
Timeloenn	250,00
AvdNr	1
EndretDato	12.1.2015

NEW

AnsNr	2
Fornavn	PETTER
Timeloenn	250,00
AvdNr	1
EndretDato	27.2.2016

OLD og NEW

To innebygde «postvariabler» OLD og NEW

- OLD → opprinnelige verdier før endring
- NEW → nye verdier slik de **vil bli** etter oppdatering i databasen
- Ved INSERT er OLD = NULL, ved DELETE er NEW = NULL

Hvordan referere til data i raden som oppdateres

- OLD.<kolonnenavn> gir gammel verdi
- NEW.<kolonnenavn> gir ny verdi

BEFORE triggere → kjøres før databasen oppdateres

- NEW-verdier er endret og tilgjengelige selv om databasen ikke er oppdatert ennå!
- Triggeren kan endre NEW-verdier **før** oppdatering i basen
- Automatisk konvertering/formatering av data som settes inn
- Automatisk oppdaterte dataverdier, f.eks. datostempel som viser når raden sist ble endret

AFTER triggere → kjøres etter at databasen er oppdatert

- Da er det **for seint** å endre NEW-verdiene.
- OLD og NEW-verdier kan fremdeles **refereres**, f.eks. for å oppdatere andre tabeller

OLD og NEW

Hva inneholder bufferne?

	OLD	NEW
INSERT	NULL i alle kolonner	Verdier i den nye raden som skal settes inn
UPDATE	Verdier slik de er/var i databasen før oppdatering	Verdier slik de vil bli i databasen etter oppdatering
DELETE	Verdier slik de er/var i databasen før sletting	NULL i alle kolonner

OLD og NEW kan bare brukes i rad-triggere!

BEFORE - AFTER

Rad-triggere

	BEFORE	AFTER
INSERT	Ny rad er ikke satt inn ennå	Ny rad er satt inn i tabellen
UPDATE	Raden er ikke oppdatert i tabellen	NEW-verdier er oppdatert i tabellen
DELETE	Raden er ikke slettet ennå	Raden er slettet

(Setnings-triggere – ikke støttet i MySQL)

	BEFORE	AFTER
INSERT	Ingen rader er satt inn ennå	Alle nye rader er satt inn i tabellen
UPDATE	Ingen rader er oppdatert ennå	Alle berørte rader er oppdatert i tabellen
DELETE	Ingen rader er slettet ennå	Alle rader som berøres av DELETE er slettet fra tabellen

Loggføring og AFTER-triggere

Hver gang noen endrer lønnen til en ansatt

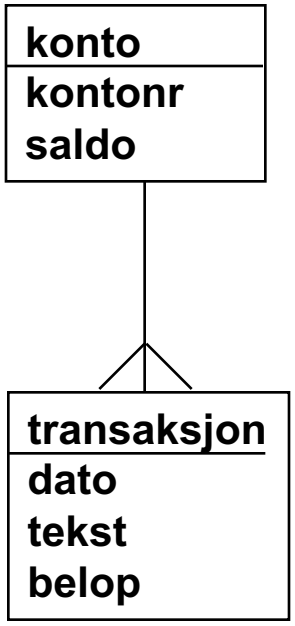
- Sett inn en rad i en egen loggtabell AnsattLogg
- (Forutsetter at tabellen er opprettet)

```
CREATE TRIGGER ansatt_a_upd_trg
  AFTER UPDATE ON ansatt
  FOR EACH ROW
BEGIN
  INSERT INTO
    AnsattLogg(AnsNr, Dato, GammelLønn, NyLønn)
  VALUES
    (NEW.AnsNr, CURDATE(), OLD.Lønn, NEW.Lønn);
END
```

Automatiske følgeoppdateringer

Oppdatere saldo på konto etter en ny transaksjon

```
CREATE OR REPLACE TRIGGER tr_trans_ins
  AFTER INSERT ON transaksjon
  FOR EACH ROW
BEGIN
  UPDATE konto
    SET konto.saldo =
      konto.saldo + NEW.belop
    WHERE konto.kontonr = NEW.kontonr;
END;
```



- Bør gjøres i AFTER-triggere når NEW-verdier er endelige
- Ikke bruk COMMIT / ROLLBACK i triggere!
- Oppdateringene bekreftes av klientprogrammet
- Ved feil skal alle endringer rulles tilbake. Det vil ikke skje hvis triggeren har gjort COMMIT!

Forretningsregler

MySQL godtar CHECK, men sjekker ikke:

```
ALTER TABLE Vare
ADD CHECK (Pris < 10000);
```

Kan bruke trigger i stedet:

```
CREATE TRIGGER vare_b_upd_trg
  BEFORE UPDATE ON Vare
  FOR EACH ROW
BEGIN
  IF NOT (NEW.Pris < 10000) THEN
    SIGNAL SQLSTATE '80000'
    SET MESSAGE_TEXT = 'For dyr vare!';
  END IF;
END
```

Beskrivelse av triggere i systemkatalogen

Metadata-kommando:

```
SHOW TRIGGERS;
```

Eller spør mot metatabellen information_schema.triggers:

```
SELECT TRIGGER_SCHEMA, TRIGGER_NAME,
       EVENT_MANIPULATION, EVENT_OBJECT_TABLE,
       ACTION_STATEMENT, ACTION_TIMING
 FROM INFORMATION_SCHEMA.triggers;
```

Eller lag en SQL-spørring som lager et SQL-skript:

```
SELECT CONCAT('DROP TRIGGER IF EXISTS ',
  TRIGGER_SCHEMA, '.', TRIGGER_NAME, ';') AS "SQL"
FROM INFORMATION_SCHEMA.TRIGGERS
```

Bruksområder for triggere

BEFORE triggere – fyres før oppdatering i basen

- Automatisk formatering av verdier
- Automatisk datostempling av rader
- Kontroll med forretningsregler
- Finmasket tilgangskontroll

AFTER triggere - fyres etter oppdatering i basen

- Automatiske oppdateringer i andre tabeller
- Loggføring av oppdateringer
- Oppdatere avledete data i andre tabeller

Brukes for operasjoner som skal gjøres «uten unntak»!

Beskrivelse av triggere i systemkatalogen

Vis informasjon om alle triggere:

```
SHOW TRIGGERS;
```

Eller spør mot metatabellen information_schema.triggers:

```
SELECT TRIGGER_SCHEMA, TRIGGER_NAME,
       EVENT_MANIPULATION, EVENT_OBJECT_TABLE,
       ACTION_STATEMENT, ACTION_TIMING
 FROM INFORMATION_SCHEMA.triggers;
```

En SQL-spørring som lager et SQL-skript som sletter alle triggere:

```
SELECT CONCAT('DROP TRIGGER IF EXISTS ',
  TRIGGER_SCHEMA, '.', TRIGGER_NAME, ';') AS "SQL"
FROM INFORMATION_SCHEMA.TRIGGERS
```