Introduction
Advanced Data Definition

Advanced Data Definition using MySQL

Kursmål

- redogöra för, analysera och kontrastera **grundläggande begrepp** och tekniska valmöjligheter som relaterar till konstruktion av databasapplikationer,
- självständigt konstruera en **fysisk datamodell** inklusive procedurer lagrade i databasen samt detaljerat redogöra för valda tekniker,
- självständigt konstruera en klientapplikation för webben som utnyttjar t**raditionella web-baserade tekniker** för att visualisera den konstruerade da- tabasen med hjälp av frågor i SQL samt detalje- rat redogöra för valda tekniker samt
- självständigt konstruera en klientapplikation som utnyttjar **MVC (Model View Controller)** ramverk för att visualisera den konstruerade da- tabasen med hjälp av frågor i SQL samt detaljerat redogöra för valda tekniker

Logisk Datamodell

Välja ett **subset** från ER modell i uppgiftsbeskrivning (ca 20%)
Välj **olika subset** än de personer som du vill diskutera uppgiften med

Överför subset av modellen till **Relationsdatamodellen.** Det viktigaste valet är för **arvshierarkier**, gör ett val av variant A-D (en eller flera tabeller för hierarkin)

Överför relationsdatamodell till **SQL**

Fysisk Datamodell

1. Obtain a set of tables from the logical database design process
2. Denormalization
3. Select data types and size for each column
4. Indexing for fast searches
5. Define constraints
6. Define Views
7. Define accounts and rights
8. Stored Procedures and Triggers

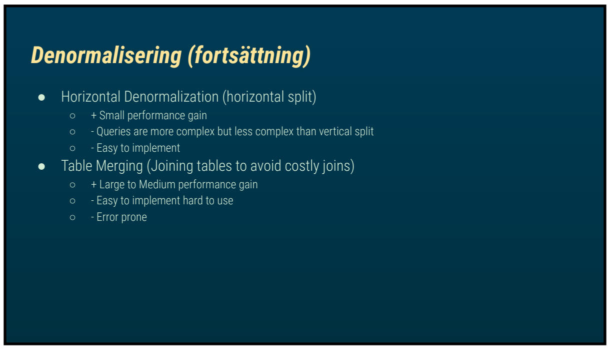
Bastabeller

```
CREATE TABLE CUSTOMER(
      CUSTNO      CHAR(6),
      SSN         CHAR(11) UNIQUE NOT NULL,
      NAME        VARCHAR(10) NOT NULL,
      REGDATE     DATETIME,
      PRIMARY KEY (CUSTNO)
) ENGINE=INNODB;

CREATE TABLE INVOICE(
      CUSTNO      CHAR(6) NOT NULL,
      INVOICENO   INTEGER NOT NULL,
      COMMENT     VARCHAR(1024),
      DATEPAID    DATETIME,
      PRIMARY KEY (CUSTNO,INVOICENO),
      FOREIGN KEY (CUSTNO) REFERENCES CUSTOMER(CUSTNO),
) ENGINE=INNODB;

CREATE TABLE INVOICEROW(
      CUSTNO      CHAR(6) NOT NULL,
      INVOICENO   INTEGER NOT NULL,
      PRODUCTNAME VARCHAR(30),
      COMPANY     VARCHAR(30),
      NUMBER      INTEGER,
      COST        REAL,
      PRIMARY KEY (CUSTNO,INVOICENO,PRODUCTNAME),
      FOREIGN KEY (CUSTNO,INVOICENO) REFERENCES INVOICE(CUSTNO,INVOICENO)
) ENGINE=INNODB;
```

SSN is är kandidatnyckel då
custno är mera lämpligt som nyckel

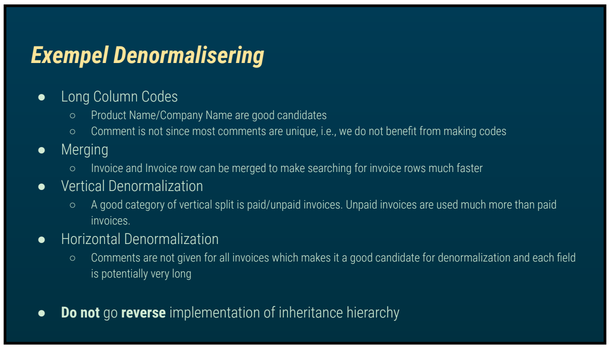
Denormalisering

- Long Column Codes (change from text to integer if text has limited range e.g. city in address)
  - + Very small performance gain
  - + Queries are only slightly more complex
  - + Very easy to implement
- Vertical Denormalization (vertical split)
  - + Major performance gain
  - - Queries are more complex
  - - Difficult to implement



Denormalisering (fortsättning)

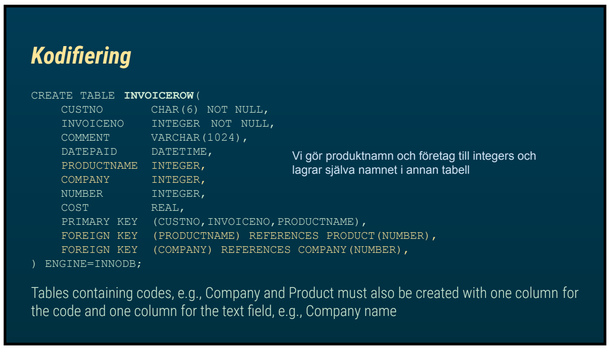
- Horizontal Denormalization (horizontal split)
  - + Small performance gain
  - - Queries are more complex but less complex than vertical split
  - - Easy to implement
- Table Merging (Joining tables to avoid costly joins)
  - + Large to Medium performance gain
  - - Easy to implement hard to use
  - - Error prone



Exempel Denormalisering

- Long Column Codes
  - Product Name/Company Name are good candidates
  - Comment is not since most comments are unique, i.e., we do not benefit from making codes
- Merging
  - Invoice and Invoice row can be merged to make searching for invoice rows much faster
- Vertical Denormalization
  - A good category of vertical split is paid/unpaid invoices. Unpaid invoices are used much more than paid invoices.
- Horizontal Denormalization
  - Comments are not given for all invoices which makes it a good candidate for denormalization and each field is potentially very long

- **Do not** go **reverse** implementation of inheritance hierarchy



Kodifiering

```
CREATE TABLE INVOICEROW(
    CUSTNO       CHAR(6) NOT NULL,
    INVOICENO    INTEGER NOT NULL,
    COMMENT      VARCHAR(1024),
    DATEPAID     DATETIME,
    PRODUCTNAME  INTEGER,
    COMPANY      INTEGER,
    NUMBER       INTEGER,
    COST         REAL,
    PRIMARY KEY  (CUSTNO,INVOICENO,PRODUCTNAME),
    FOREIGN KEY  (PRODUCTNAME) REFERENCES PRODUCT(NUMBER),
    FOREIGN KEY  (COMPANY) REFERENCES COMPANY(NUMBER),
) ENGINE=INNODB;
```

Vi gör produktnamn och företag till integers och lagrar själva namnet i annan tabell

Tables containing codes, e.g., Company and Product must also be created with one column for the code and one column for the text field, e.g., Company name

Merging

```
CREATE TABLE INVOICEROW(
     CUSTNO       CHAR(6) NOT NULL,
     INVOICENO    INTEGER  NOT NULL,
     COMMENT      VARCHAR(1024),
     DATEPAID     DATETIME,
     PRODUCTNAME  VARCHAR(30),
     COMPANY      VARCHAR(30),
     NUMBER       INTEGER,
     COST         REAL,
     PRIMARY KEY  (CUSTNO,INVOICENO,PRODUCTNAME),
     FOREIGN KEY  (CUSTNO) REFERENCES CUSTOMER(CUSTNO),
) ENGINE=INNODB;
```

Vi tar bort tabellen invoice och flyttar de unika kolumnerna som invoice har in i invoicerow

Comment och Datepaid är då redundanta

(Ordering of steps is unique for each application or set of tables)

Horizontal Denormalization

```
CREATE TABLE INVOICEROW(
     CUSTNO       CHAR(6) NOT NULL,
     INVOICENO    INTEGER NOT NULL,
     DATEPAID     DATETIME,
     PRODUCTNAME  INTEGER,
     COMPANY      INTEGER,
     NUMBER       INTEGER,
     COST         REAL,
     PRIMARY KEY  (CUSTNO,INVOICENO,PRODUCTNAME),
     FOREIGN KEY  (COMPANY) REFERENCES COMPANY(NUMBER),
     FOREIGN KEY  (PRODUCTNAME) REFERENCES PRODUCT(NUMBER),
) ENGINE=INNODB;

CREATE TABLE INVOICEROWCOMMENT(
     CUSTNO       CHAR(6) NOT NULL,
     INVOICENO    INTEGER NOT NULL,
     PRODUCTNAME  INTEGER,
     COMMENT      VARCHAR(1024),
     PRIMARY KEY  (CUSTNO,INVOICENO,PRODUCTNAME),
     FOREIGN KEY  (CUSTNO,INVOICENO,PRODUCTNAME) REFERENCES INVOICEROW,
) ENGINE=INNODB;
```

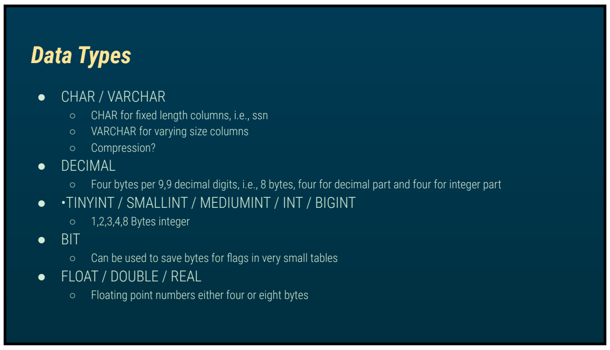
Comment (och/eller andra långa attribut) lyfts ut och vi ger samma nyckel som bastabellen

Vertical Denormalization

```
CREATE TABLE INVOICEROW(
        CUSTNO                  CHAR(6) NOT NULL,
        INVOICENO               INTEGER NOT NULL,
        DATEPAID                DATETIME,
        PRODUCTNAME             INTEGER,
        COMPANY                 INTEGER,
        NUMBER                  INTEGER,
        COST                    REAL,
        PRIMARY KEY             (CUSTNO,INVOICENO,PRODUCTNAME),
        FOREIGN KEY             (COMPANY) REFERENCES COMPANY(NUMBER),
        FOREIGN KEY             (PRODUCTNAME) REFERENCES PRODUCT(NUMBER),
) ENGINE=INNODB;

CREATE TABLE PAIDINVOICEROW(
        CUSTNO                  CHAR(6) NOT NULL,
        INVOICENO               INTEGER NOT NULL,
        DATEPAID                DATETIME,
        PRODUCTNAME             INTEGER,
        COMPANY                 INTEGER,
        NUMBER                  INTEGER,
        COST                    REAL,
        PRIMARY KEY             (CUSTNO,INVOICENO,PRODUCTNAME),
        FOREIGN KEY             (COMPANY) REFERENCES COMPANY(NUMBER),
        FOREIGN KEY             (PRODUCTNAME) REFERENCES PRODUCT(NUMBER),
) ENGINE=INNODB;

CREATE TABLE INVOICEROWCOMMENT(
        CUSTNO                  CHAR(6) NOT NULL,
        INVOICENO               INTEGER NOT NULL,
        PRODUCTNAME             INTEGER,
        COMMENT                 VARCHAR(1024),
        PRIMARY KEY             (CUSTNO,INVOICENO,PRODUCTNAME),
        FOREIGN KEY             (CUSTNO,INVOICENO,PRODUCTNAME) REFERENCES INVOICEROW,
) ENGINE=INNODB;

CREATE TABLE PAIDINVOICEROWCOMMENT(
        CUSTNO                  CHAR(6) NOT NULL,
        INVOICENO               INTEGER NOT NULL,
        PRODUCTNAME             INTEGER,
        COMMENT                 VARCHAR(1024),
        PRIMARY KEY             (CUSTNO,INVOICENO,PRODUCTNAME),
        FOREIGN KEY             (CUSTNO,INVOICENO,PRODUCTNAME) REFERENCES PAIDINVOICEROW,
) ENGINE=INNODB;
```

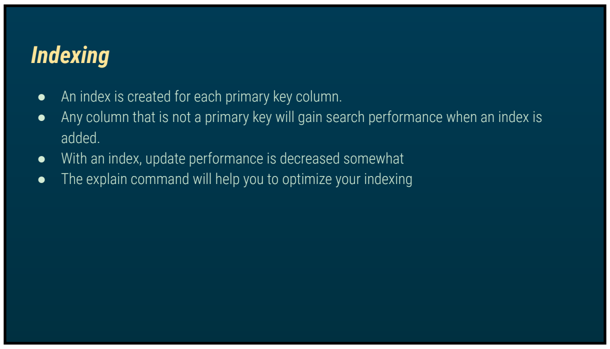
Kopiera tabellen vi vill dela upp och alla främmande nyckeltabeller som refererar den tabellen (i detta fallet comment)

Mycket liten extra vinst om vi splittrar både horizontellt och vertikalt på samma tabell som i detta fallet



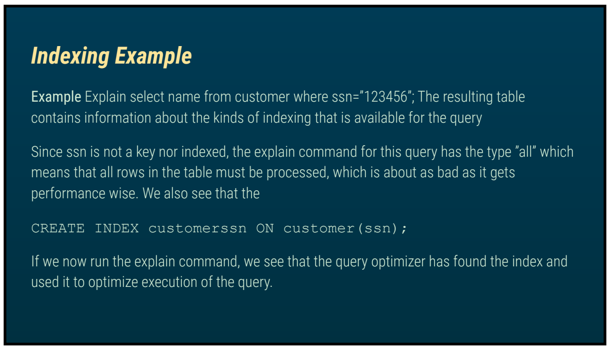
Data Types

- CHAR / VARCHAR
  - CHAR for fixed length columns, i.e., ssn
  - VARCHAR for varying size columns
  - Compression?
- DECIMAL
  - Four bytes per 9,9 decimal digits, i.e., 8 bytes, four for decimal part and four for integer part
- •TINYINT / SMALLINT / MEDIUMINT / INT / BIGINT
  - 1,2,3,4,8 Bytes integer
- BIT
  - Can be used to save bytes for flags in very small tables
- FLOAT / DOUBLE / REAL
  - Floating point numbers either four or eight bytes



Indexing

- An index is created for each primary key column.
- Any column that is not a primary key will gain search performance when an index is added.
- With an index, update performance is decreased somewhat
- The explain command will help you to optimize your indexing



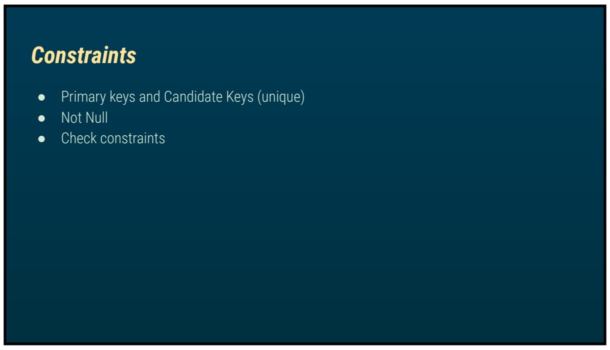
Indexing Example

Example Explain select name from customer where ssn="123456"; The resulting table contains information about the kinds of indexing that is available for the query

Since ssn is not a key nor indexed, the explain command for this query has the type "all" which means that all rows in the table must be processed, which is about as bad as it gets performance wise. We also see that the

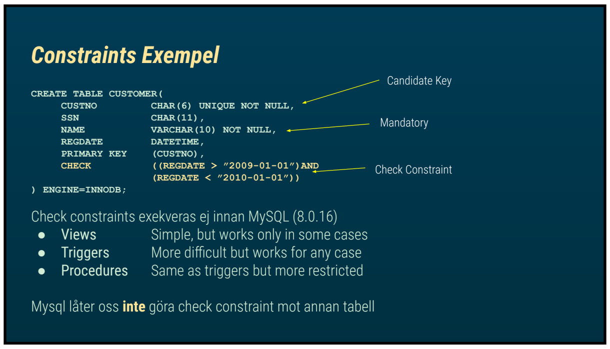
```
CREATE INDEX customerssn ON customer(ssn);
```

If we now run the explain command, we see that the query optimizer has found the index and used it to optimize execution of the query.



Constraints

- Primary keys and Candidate Keys (unique)
- Not Null
- Check constraints



Constraints Exempel

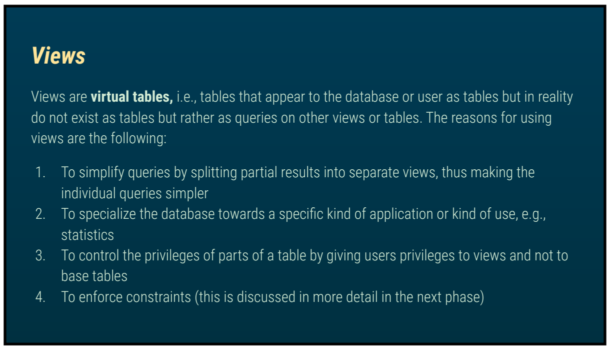
```
CREATE TABLE CUSTOMER(
     CUSTNO         CHAR(6) UNIQUE NOT NULL,
     SSN            CHAR(11),
     NAME           VARCHAR(10) NOT NULL,
     REGDATE        DATETIME,
     PRIMARY KEY    (CUSTNO),
     CHECK          ((REGDATE > ”2009-01-01”)AND
                    (REGDATE < ”2010-01-01”))
) ENGINE=INNODB;
```

Candidate Key
Mandatory
Check Constraint

Check constraints exekveras ej innan MySQL (8.0.16)

● Views	Simple, but works only in some cases
● Triggers	More difficult but works for any case
● Procedures	Same as triggers but more restricted

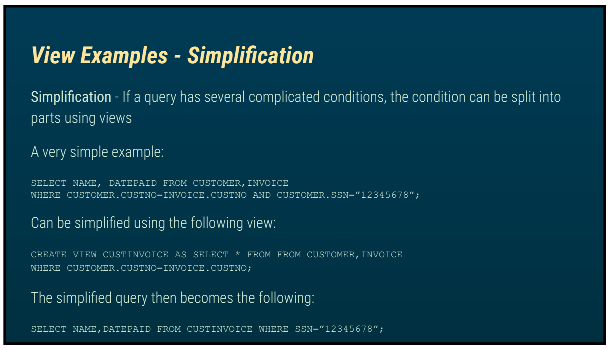
Mysql låter oss **inte** göra check constraint mot annan tabell



Views

Views are **virtual tables,** i.e., tables that appear to the database or user as tables but in reality do not exist as tables but rather as queries on other views or tables. The reasons for using views are the following:

1. To simplify queries by splitting partial results into separate views, thus making the individual queries simpler
2. To specialize the database towards a specific kind of application or kind of use, e.g., statistics
3. To control the privileges of parts of a table by giving users privileges to views and not to base tables
4. To enforce constraints (this is discussed in more detail in the next phase)



View Examples - Simplification

Simplification - If a query has several complicated conditions, the condition can be split into parts using views

A very simple example:

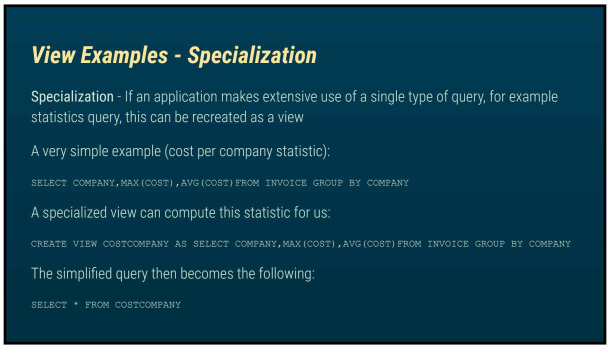
```
SELECT NAME, DATEPAID FROM CUSTOMER,INVOICE
WHERE CUSTOMER.CUSTNO=INVOICE.CUSTNO AND CUSTOMER.SSN="12345678";
```

Can be simplified using the following view:

```
CREATE VIEW CUSTINVOICE AS SELECT * FROM FROM CUSTOMER,INVOICE
WHERE CUSTOMER.CUSTNO=INVOICE.CUSTNO;
```

The simplified query then becomes the following:

```
SELECT NAME,DATEPAID FROM CUSTINVOICE WHERE SSN="12345678";
```



View Examples - Specialization

Specialization - If an application makes extensive use of a single type of query, for example statistics query, this can be recreated as a view

A very simple example (cost per company statistic):

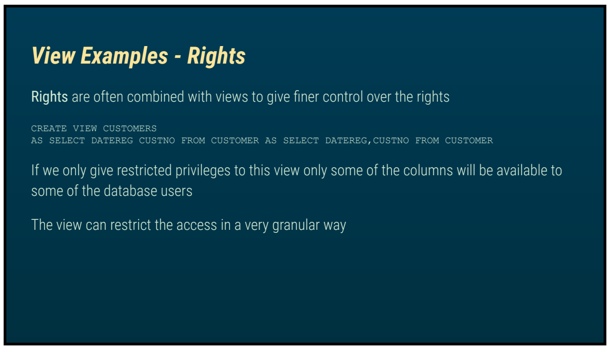
```
SELECT COMPANY,MAX(COST),AVG(COST)FROM INVOICE GROUP BY COMPANY
```

A specialized view can compute this statistic for us:

```
CREATE VIEW COSTCOMPANY AS SELECT COMPANY,MAX(COST),AVG(COST)FROM INVOICE GROUP BY COMPANY
```

The simplified query then becomes the following:

```
SELECT * FROM COSTCOMPANY
```



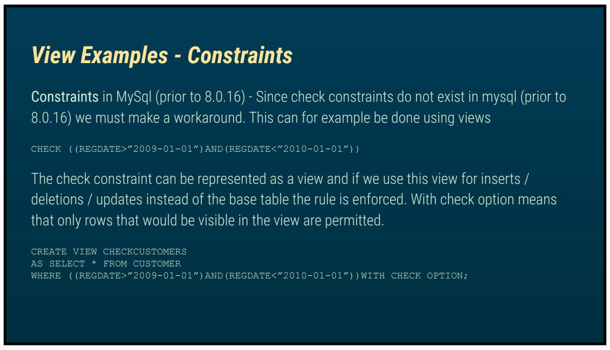
View Examples - Rights

Rights are often combined with views to give finer control over the rights

```
CREATE VIEW CUSTOMERS
AS SELECT DATEREG CUSTNO FROM CUSTOMER AS SELECT DATEREG,CUSTNO FROM CUSTOMER
```

If we only give restricted privileges to this view only some of the columns will be available to some of the database users

The view can restrict the access in a very granular way



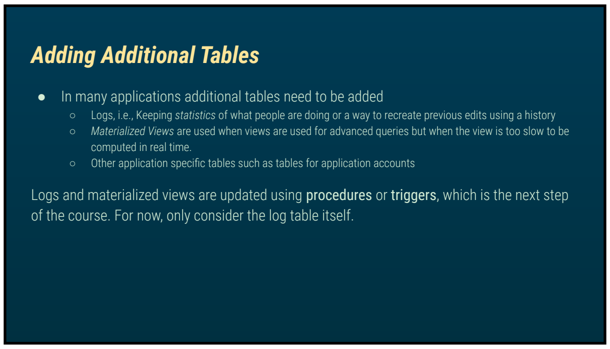
View Examples - Constraints

Constraints in MySql (prior to 8.0.16) - Since check constraints do not exist in mysql (prior to 8.0.16) we must make a workaround. This can for example be done using views

```
CHECK ((REGDATE>”2009-01-01”)AND(REGDATE<”2010-01-01”))
```

The check constraint can be represented as a view and if we use this view for inserts / deletions / updates instead of the base table the rule is enforced. With check option means that only rows that would be visible in the view are permitted.

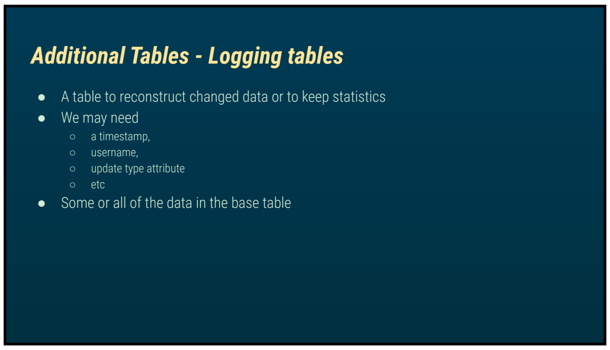
```
CREATE VIEW CHECKCUSTOMERS
AS SELECT * FROM CUSTOMER
WHERE ((REGDATE>”2009-01-01”)AND(REGDATE<”2010-01-01”))WITH CHECK OPTION;
```



Adding Additional Tables

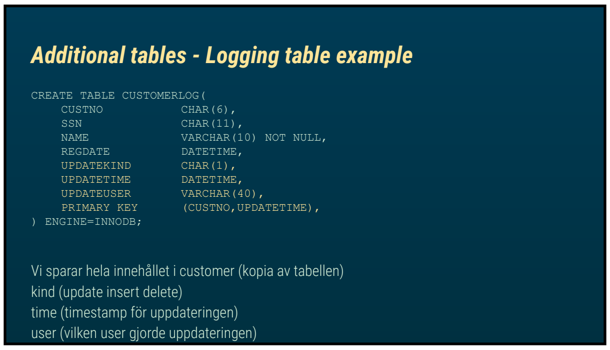
- In many applications additional tables need to be added
  - Logs, i.e., Keeping *statistics* of what people are doing or a way to recreate previous edits using a history
  - *Materialized Views* are used when views are used for advanced queries but when the view is too slow to be computed in real time.
  - Other application specific tables such as tables for application accounts

Logs and materialized views are updated using **procedures** or **triggers**, which is the next step of the course. For now, only consider the log table itself.



Additional Tables - Logging tables

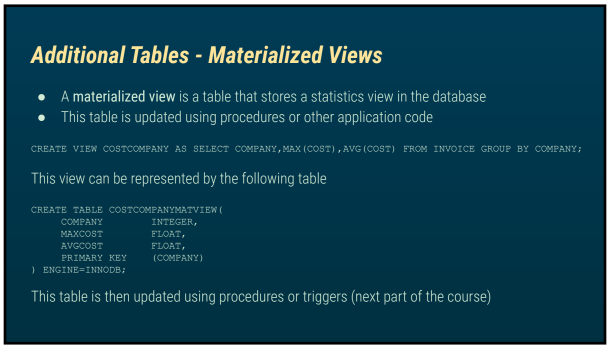
- A table to reconstruct changed data or to keep statistics
- We may need
  - a timestamp,
  - username,
  - update type attribute
  - etc
- Some or all of the data in the base table



Additional tables - Logging table example

```
CREATE TABLE CUSTOMERLOG(
     CUSTNO            CHAR(6),
     SSN               CHAR(11),
     NAME              VARCHAR(10) NOT NULL,
     REGDATE           DATETIME,
     UPDATEKIND        CHAR(1),
     UPDATETIME        DATETIME,
     UPDATEUSER        VARCHAR(40),
     PRIMARY KEY       (CUSTNO,UPDATETIME),
) ENGINE=INNODB;
```

Vi sparar hela innehållet i customer (kopia av tabellen)
kind (update insert delete)
time (timestamp för uppdateringen)
user (vilken user gjorde uppdateringen)



Additional Tables - Materialized Views

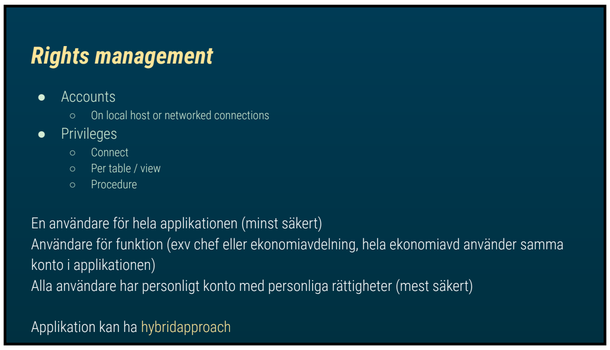
- A **materialized view** is a table that stores a statistics view in the database
- This table is updated using procedures or other application code

```
CREATE VIEW COSTCOMPANY AS SELECT COMPANY,MAX(COST),AVG(COST) FROM INVOICE GROUP BY COMPANY;
```

This view can be represented by the following table

```
CREATE TABLE COSTCOMPANYMATVIEW(
     COMPANY        INTEGER,
     MAXCOST        FLOAT,
     AVGCOST        FLOAT,
     PRIMARY KEY    (COMPANY)
) ENGINE=INNODB;
```

This table is then updated using procedures or triggers (next part of the course)

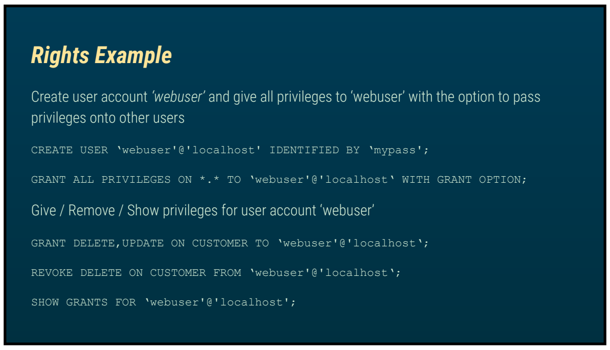


Rights management

- Accounts
  - On local host or networked connections
- Privileges
  - Connect
  - Per table / view
  - Procedure

En användare för hela applikationen (minst säkert)
Användare för funktion (exv chef eller ekonomiavdelning, hela ekonomiavd använder samma konto i applikationen)
Alla användare har personligt konto med personliga rättigheter (mest säkert)

Applikation kan ha hybridapproach



Rights Example

Create user account *‘webuser’* and give all privileges to ‘webuser’ with the option to pass privileges onto other users

```
CREATE USER ‘webuser'@'localhost' IDENTIFIED BY ‘mypass';

GRANT ALL PRIVILEGES ON *.* TO ‘webuser'@'localhost‘ WITH GRANT OPTION;
```

Give / Remove / Show privileges for user account ‘webuser’

```
GRANT DELETE,UPDATE ON CUSTOMER TO ‘webuser'@'localhost‘;

REVOKE DELETE ON CUSTOMER FROM ‘webuser'@'localhost‘;

SHOW GRANTS FOR ‘webuser'@'localhost';
```