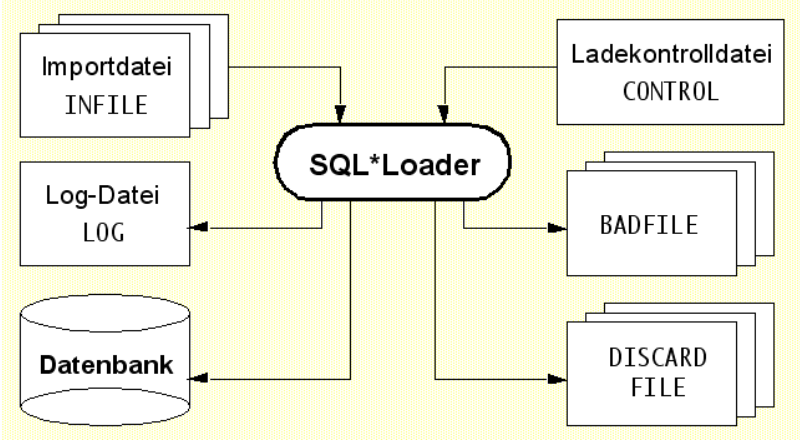
Oracle SQL*Loader

Der **Oracle SQL*Loader** ist Programm zum **Einladen von Daten** aus Dateien in die Datenbank.

Konzept



- **Bad File:** Datensätze, die vom SQL*Loader oder Oracle zurückgewiesen worden sind
- **Discard File:** optional, Datensätze, die evtl. angegebenen Filterkriterien nicht gehorchten
- **Log File:** Infos über den Lade-Prozess

Kontrolldateien

Kontrolldateien (**Control Files**) beschreiben den Aufbau der Eingabedateien

- typisches Dateikürzel: `ctl`

Kontrolldatei für Dateien mit fester Datensatzlänge

```
LOAD DATA
INFILE 'standorte.txt' "fix10"
BADFILE 'standorte.bad'
INTO TABLE HatStandortIn
REPLACE
( hs         position(1-1),
  gemeinde   position(2-8)
)
```

- APPEND = Daten zusätzlich der Tabelle hinzufügen
- INSERT = Tabelle muss leer sein
- REPLACE = alle vorhandenen Datensätze werden gelöscht
- TRUNCATE = REPLACE. Es setzt allerdings voraus, dass keine Integritätsbedingungen für die zu löschenden Datensätze aktiviert sind => schneller

- Datenbeispiel:

```
33241001
13403000
23403000
23402000
23405000
```

Kontrolldatei für Dateien mit variabler Datensatzlänge

```
LOAD DATA
INFILE 'kreise.txt'
```

```
BADFILE 'kreise.bad'
INTO TABLE Kreise
REPLACE
FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '"'
( kkz,
  name,
  qkm TERMINATED BY ' ',
  kreisstadt )
```

- Datenbeispiel:

```
3403;"Oldenburg Stadt";102,97 3403000
3451;Ammerland;728,16 3451007
3404;Osnabrück Stadt;119,8 3404000
3459;Osnabrück Land;2121,56 3404000
```

Am besten nimmt man Vorlagen und ändert diese gemäß nach den eigenen Ansprüchen ab!

Aufruf

```
SQLLDR USERID=<Benutzer>@<Dienstname>
       CONTROL=standorte.ctl LOG=standorte.log ERRORS=50
```

Oracle SQL*Loader für Geodaten

Der **Oracle SQL*Loader** ist auch zum Laden vom SDO_GEOMETRY-Objekten geeignet.

Import von Punktdaten

Es ist meist sinnvoll, die Importdaten erst in ein temporäre Tabelle zu laden und dann erst über UPDATE-Anweisungen in die eigentliche Zieltabelle zu überführen.

Definition der temporären Tabelle:

```
CREATE TABLE GeoTemp (
  id     DECIMAL(8),                    -- ID
  geo    MDSYS.SDO_GEOMETRY,            -- Objektgeometrie
  CONSTRAINT pk_geotemp PRIMARY KEY(id) -- Primärschlüssel
);
```

Kontrolldatei

```
LOAD DATA INFILE 'centrum.txt'
TRUNCATE
INTO TABLE GeoTemp
FIELDS TERMINATED BY ';'
TRAILING NULLCOLS
(  id,
   geo  COLUMN OBJECT
   (  sdo_gtype,
      sdo_srid,
      sdo_point  COLUMN OBJECT  (x, y, z)
   )
)
```

- TRAILING NULLCOLS = nicht in der Datei enthaltende letzte Attributwerte eines Datensatzes werden auf NULL gesetzt.
- COLUMN OBJECT = Objekt

Datenbeispiel:

```
3241001;2001;4326;9,7358;52,3803
3403000;2001;4326;8,2275;53,1375;7
3451002;2001;4326;7,9981;53,1844
3451007;2001;4326;7,9256;53,2581
3402000;2001;4326;7,2072;53,3686
3404000;2001;4326;8,0410;52,2772
3405000;2001;4326;8,1081;53,5397
4011000;2001;4326;8,8075;53,0761
```

Import von komplexen Geometrien

```
-- Tabelle anlegen:
CREATE TABLE Ortsnetze (
  vorwahl VARCHAR(6),      -- Vorwahlnummer
  name VARCHAR(80),        -- Name des Ortsnetzes
  gebiet  SDO_GEOMETRY,    -- Gebiet des Ortsnetzes
  CONSTRAINT pk_vorwahlgebiete PRIMARY KEY(vorwahl)
);

-- Metadaten einfügen:
INSERT INTO USER_SDO_GEOM_METADATA
VALUES ('ORTSNETZE','GEBIET', SDO_DIM_ARRAY(
```

```
  SDO_DIM_ELEMENT('X',-180,180,1),
  SDO_DIM_ELEMENT('Y',-90,90,1)
) , 4326);
COMMIT;
```

```
LOAD DATA
INFILE 'vorwahl.txt'
TRUNCATE
CONTINUEIF NEXT(1:1) = '#'
INTO TABLE Ortsnetze
FIELDS TERMINATED BY ';'
TRAILING NULLCOLS
(  vorwahl,
   name,
   gebiet  COLUMN OBJECT
   (  sdo_gtype,
      sdo_elem_info  VARRAY  TERMINATED BY '|' (elements),
      sdo_ordinates  VARRAY  TERMINATED BY 'end' (ordinates)
   ) )
```

- CONTINUEIF NEXT(1:1)='#' = Fortsetzungszeichen für Datensätze, die sich über mehrere Zeilen erstrecken.
- VARRAY = Feld; Angabe einer Trennzeichenkette und eines (beliebigen) Namens für die Feldelemente erforderlich.

Datenbeispiel:

```
04921;Emden Stadt;2003;4326; 1;1003;1|7,13056;53,39298;
#7,13300;53,38763;7,11723;53,37316;7,11493;53,37183;
#7,11429;53,37051;7,11490;53,36988;7,11376;53,36907;
#7,11054;53,36794;7,10843;53,36876;7,10505;53,36703;
  -- und so weiter ...
#7,14302;53,39435;7,14229;53,39609;7,14147;53,39751;
#7,13268;53,39391;7,13056;53,39298 end
```

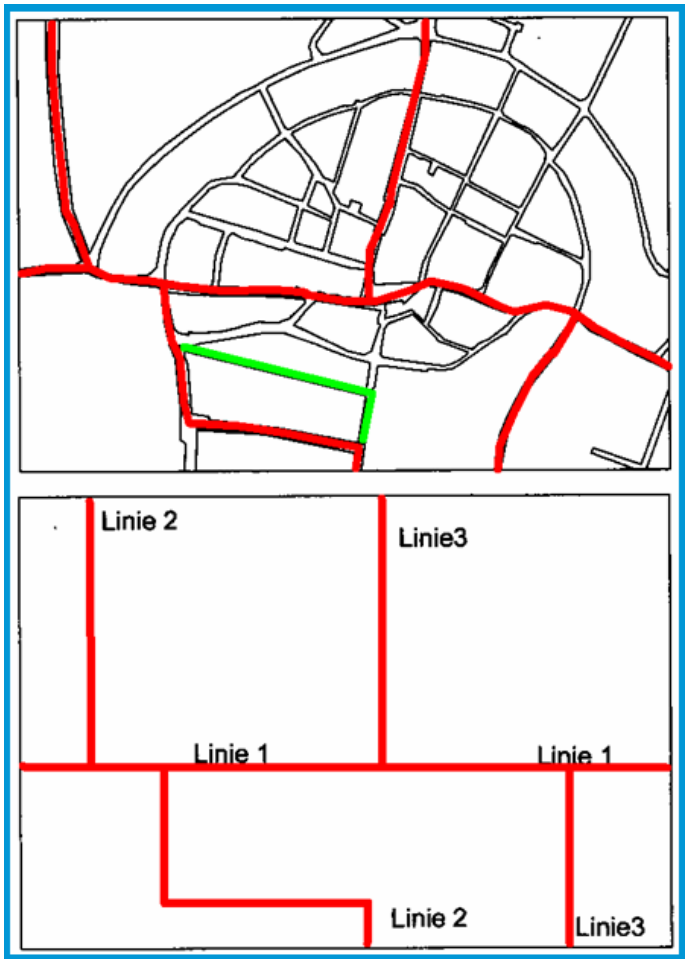
Räumliche Netzwerkdatenbanken

Einen wichtigen **Spezialfall** von Geodatenbanken stellen räumliche Netzwerkdatenbanken dar:

> **Räumliche Netzwerkdatenbanken** kombinieren die Repräsentation der **Netzwerktopologie** mit **Lageinformationen**.

- Beispiele:
  - Straßennetze (z.B. für Straßenbanken und andere Anwendungen der Verkehrstelematik)
  - Telekommunikationsnetze (z.B. für Telefon- und Breitbandkabelnetze)
  - Ver- und Entsorgungsnetze(z.B. für Strom-, Wasser-, Kanal- und Gasnetze bei Energieversorgungsunternehmen) (**NIS**)

Netzwerkdatenbanken erfordern die Speicherung der **Topologie**!



Topologische Datenmodelle

Topologische Eigenschaften können

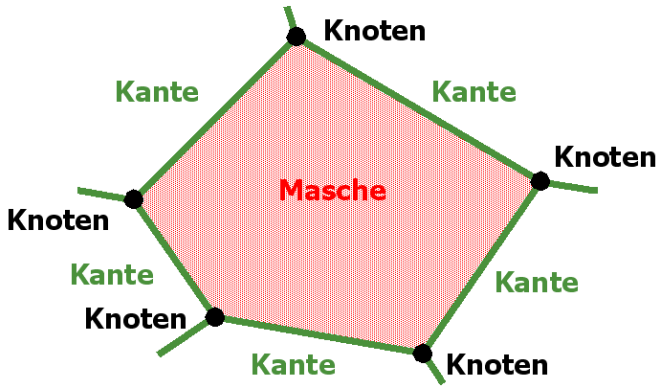
- aus den geometrischen Eigenschaften **abgeleitet** werden oder
- explizit durch ein **Datenmodell** repräsentiert werden.

? Vor- / Nachteile?

Topologische Datenmodelle beschreiben die Topologie mittels **topologischer Primitive**.

Topologische Primitive (bis 2D):

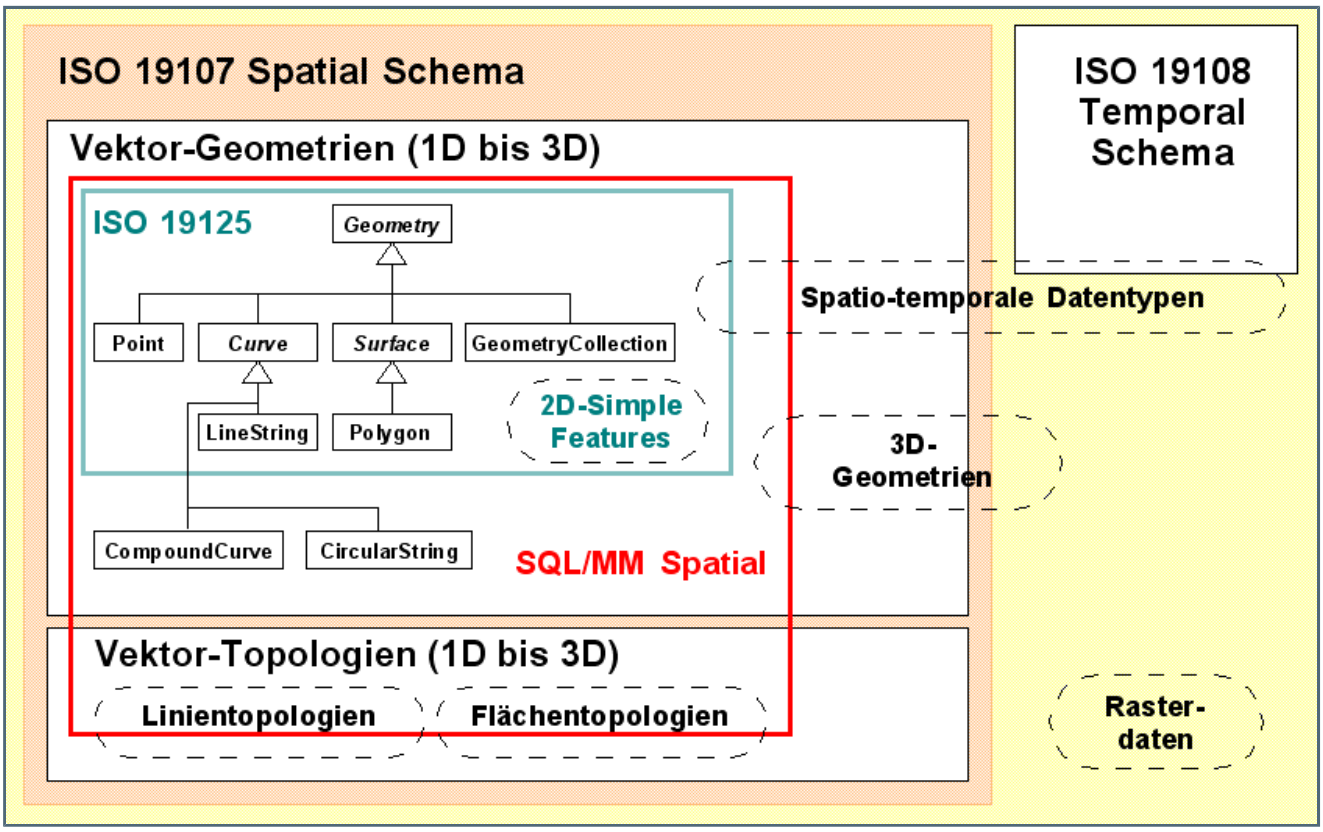
- **Knoten** (engl. **Nodes** oder **Vertices** (sing. **Vertex**)) für **Punkte** im Raum.
- **Kanten** (engl. **Edges**; auch: **Links**) für (linienhafte) **Verbindungen** zwischen Knoten.
- **Maschen** (engl. **Faces**) für **Flächen**.



Ein **Netzwerk** ist ein **linienhaftes Knoten-Kanten-Modell**, bei dem auf Maschen verzichtet wird.

Standards für räumliche und topologische Datenbanken

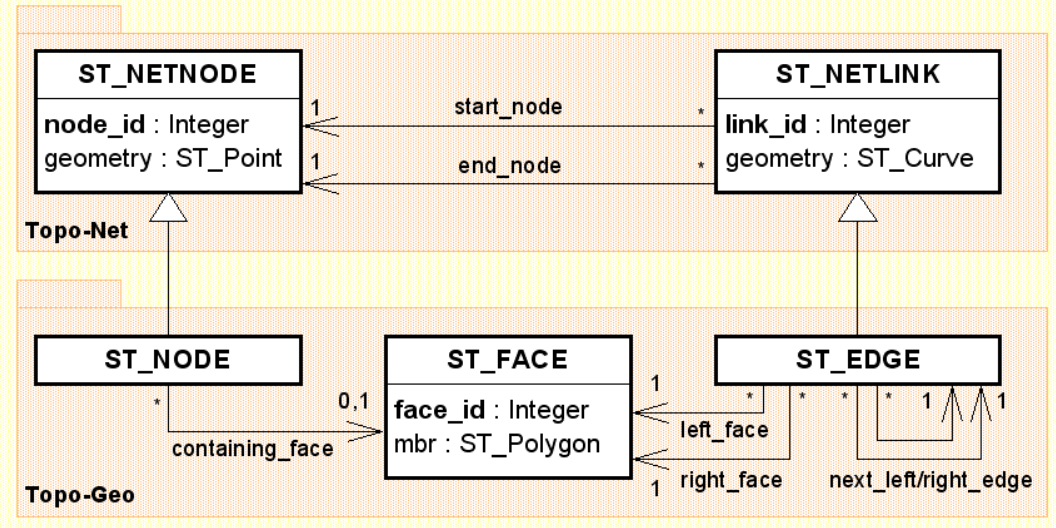
Übersicht



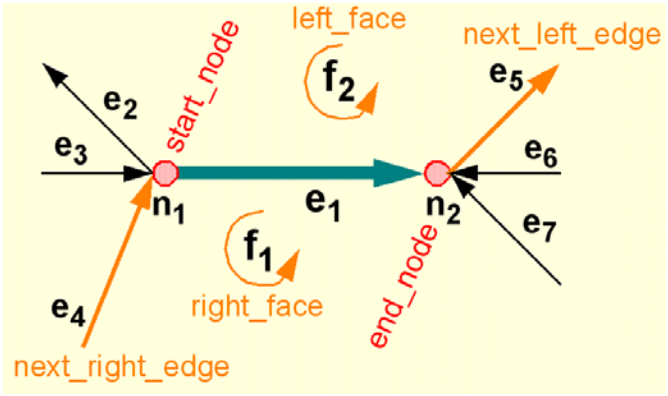
Topologie in SQL/MM

ISO 13249-3:2006 SQL/MM Spatial enthält seit der 3. Edition von 2006 ein **Netzwerk-** und ein **Topologiedatenbankschema**.

Aufbau



Beispiel

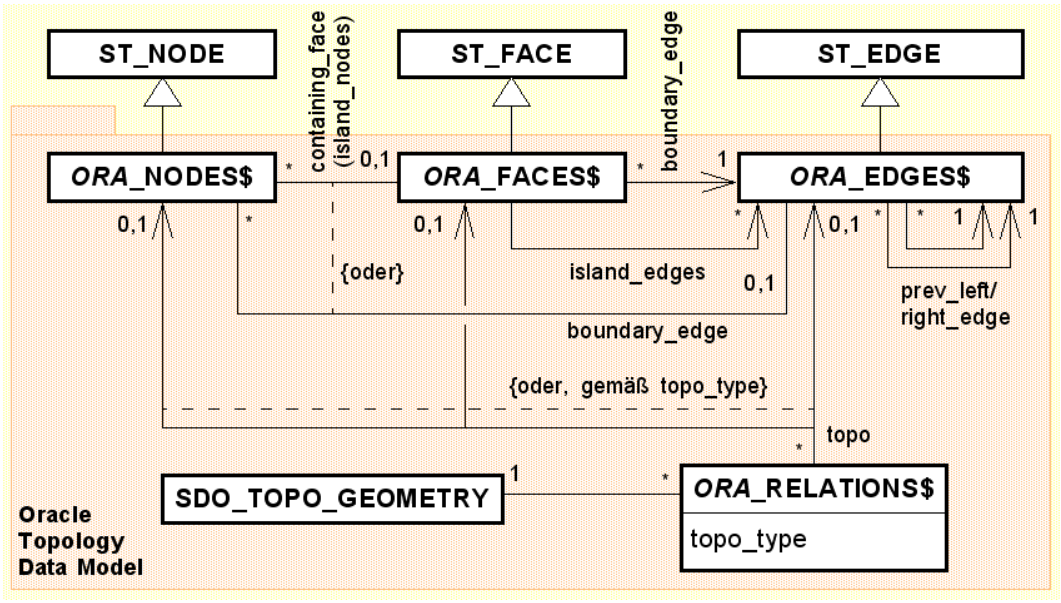


Der Standard definiert keine spezifischen **Analysefunktionen** für die topologische Datenbankschemata.

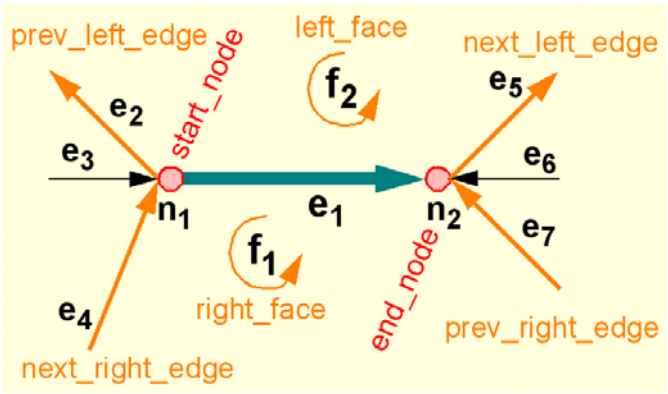
Topologie in Oracle

Oracle enthält seit der Version 10 ein **Netzwerk-** und ein **Topologiedatenbankschema**.

Aufbau

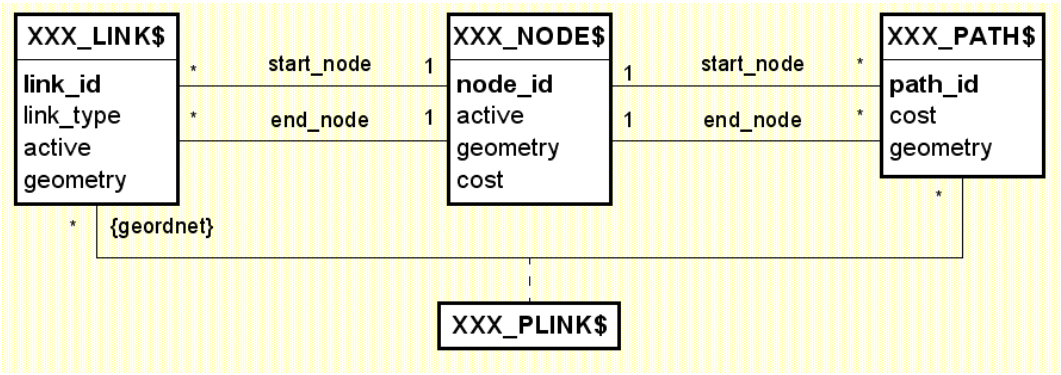


Beispiel



Topologie in Oracle

Netzwerktabellen



Beispiel

```
Name        Typ
----------- --------------
node_id     NUMBER                -- Knoten-Identifier
node_name   VARCHAR2(200)         -- Benutzerdef. Knotenname
node_type   VARCHAR2(200)         -- Benutzerdef. Knotenklasse/
active      VARCHAR2(1)           -- 'Y' für "sichtbare" Knoten
geometry    SDO_GEOMETRY          -- Punktgeometrie des Knotens
cost        NUMBER                -- Kosten beim Passieren
```

```
Name          Null?    Typ
------------- -------- -------------
link_id       NOT NULL NUMBER          -- Kanten-Identifier
link_name              VARCHAR2(200)   -- Benutzerdef. Kantenanme
start_node_id NOT NULL NUMBER          -- ID des Anfangsknotens
end_node_id   NOT NULL NUMBER          -- ID des Endknotens
link_type              VARCHAR2(200)   -- Benutzerdef. Kantenklasse
active                 VARCHAR2(1)     -- 'Y' für "sichtbare" Kanten
geometry               SDO_GEOMETRY    -- Liniengeometrie der Kante
cost                   NUMBER          -- Kosten beim Ablaufen
```

```
-- Netzwerk anlegen:
DECLARE
BEGIN
  SDO_NET.CREATE_SDO_NETWORK('OldenburgNetz', 1, FALSE, FALSE);
END;
/
```

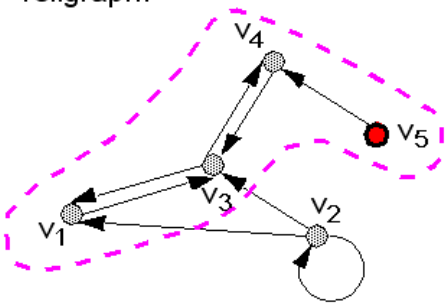
Netzwerkanalyse

Mit Hilfe eines Graphen lassen sich eine Reihe typischer **Netzwerkanalyseoperationen** durchführen.

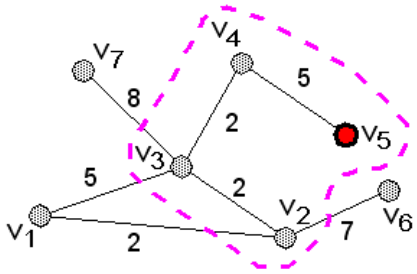
Die **Erreichbarkeit eines Knoten** d von einem Knoten s entspricht der **Existenz (mindestens) eines Weges** von s nach d.

- Ähnlich: die **Bestimmung aller Knoten**, die von s aus erreicht werden können
  = der **größte zusammenhängende (Teil-) Graph**, der s umfasst.

von v_5 erreichbarer Teilgraph:



von v_5 mit maximalen Kosten von 10 erreichbar:



Eine **Netzwerkverfolgung** bestimmt alle Knoten oder Kanten, die vom Knoten s unter Einhaltung einer maximalen Weglänge erreicht werden können.

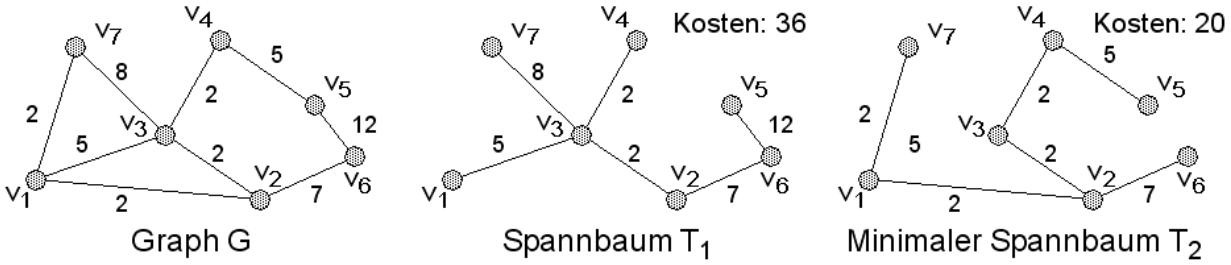
- Betrachtet man gleichzeitig mehrere Standorte, muss ggf. die Netzwerkverfolgung auf **halben Weg** zwischen zwei Standorten enden.

Der **kürzeste Weg** (engl. **Shortest Path**) zwischen den Knoten s und d ist der Weg, der die **geringste Länge** aufweist.

- Variante: **Berechnung aller Wege** zwischen zwei Knoten, die unterhalb einer vorgegebenen Kostengrenze bleiben.

Beim **Spannbaumproblem** (engl. **Spanning Tree Problem**) wird für einen ungerichten, zusammenhängenden Graphen G ein Baum T bestimmt, der alle Knoten des Graphen G verbindet, wobei eine Teilmenge der Kanten von G die Kanten von T bilden.

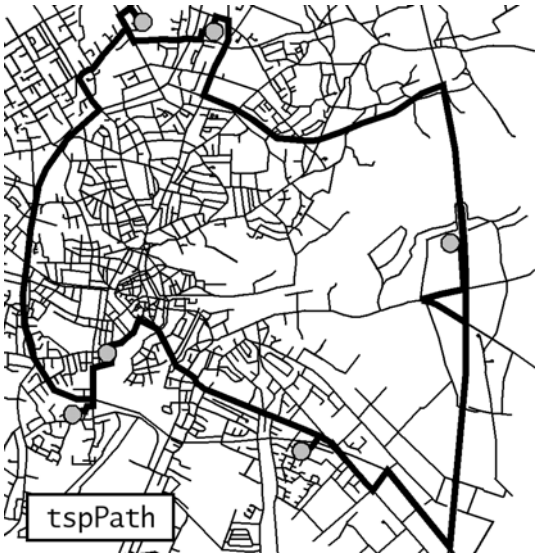
- T ist also ein zusammenhängender Graph, bei dem es zwischen zwei Knoten immer genau einen Weg gibt.



- Beim **minimalen Spannbaum** (engl. **Minimum Cost Spanning Tree**) ist die Summe der Kosten der Kanten **minimal**.

Beim **Problem des Handlungsreisenden** (engl. **Traveling Salesman Problem**) wird in einem Graphen G der kürzeste Weg gesucht, der alle Knoten besucht und wieder zum Ausgangsknoten zurückkehrt.

- Aufgrund der exponentiellen Laufzeit der optimalen Lösung werden oft Näherungslösungen verwendet, die maximal um einen vorgegebenen Prozentsatz von der optimalen Lösung abweichen.



Oracle

- Oracle unterstützt die genannten Netzwerkanalysefunktionen über Java- oder PL/SQL-Aufrufe.

Anwendungsbeispiel

Wegedatenbank

- Basis-Netznetzwerk mit SDO_GEOMETRY-Geometrien
- Lineares Bezugssystem für Kilometrierung
- Segmente mit Bezug zum Basis-Netzwerk und ggf. eigener Kilometrierung

Beispiel

```
-- Netzwerk vereinbaren
-- --------------------

DECLARE
BEGIN
  SDO_NET.CREATE_SDO_NETWORK('Basis', 1, FALSE, TRUE);
END;
/

-- Knotentabelle
-- -------------

CREATE TABLE BASIS_NODES$ (
  node_id    NUMBER,                        -- Knoten-Identifier
  node_name  VARCHAR2(200),                 -- Benutzerdefinierte Knotenbezeichnung
  node_type  VARCHAR2(200),                 -- Benutzerdefinierte Knotenklasse/-typ
  active     VARCHAR2(1)    DEFAULT 'Y',    -- 'Y' für "sichtbare" Knoten
  geometry   SDO_GEOMETRY,                  -- Punktgeometrie des Knotens
  cost       NUMBER         DEFAULT 0,      -- Kosten beim Passieren des Knotens
  CONSTRAINT basisnodes_pk PRIMARY KEY (node_id)
);

INSERT INTO User_SDO_Geom_Metadata
VALUES ('BASIS_NODES$', 'GEOMETRY' ,
        MDSYS.SDO_DIM_ARRAY( MDSYS.SDO_DIM_ELEMENT('LAENGE', -180, 180, 0.000005),
        MDSYS.SDO_DIM_ELEMENT('BREITE',  -90, 90, 0.000005)), 4326 );
COMMIT;

CREATE INDEX Basis_Nodes_geom_ix
ON BASIS_NODES$(geometry)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;


-- Kantentabelle
-- -------------

CREATE TABLE BASIS_LINKS$ (
  link_id        NUMBER,                     -- Kanten-Identifier
  link_name      VARCHAR2(200),              -- Benutzerdef. Kantenbezeichnung
  start_node_id  NUMBER         NOT NULL,    -- ID des Anfangsknotens
  end_node_id    NUMBER         NOT NULL,    -- ID des Endknotens
  link_type      VARCHAR2(200),              -- Benutzerdef. Kantenklasse/-typ
  active         VARCHAR2(1)    DEFAULT 'Y', -- 'Y' für "sichtbare" Kanten
  geometry       SDO_GEOMETRY,               -- Liniengeometrie der Kante
  cost           NUMBER         DEFAULT 0,   -- Kosten beim Ablaufen der Kante
  CONSTRAINT basislinks_pk PRIMARY KEY (link_id),
  CONSTRAINT fk_basislinks_sn
    FOREIGN KEY (start_node_id) REFERENCES BASIS_NODES$(node_id),
  CONSTRAINT fk_basislinks_en
    FOREIGN KEY (end_node_id) REFERENCES BASIS_NODES$(node_id)
);

INSERT INTO User_SDO_Geom_Metadata
VALUES ('BASIS_LINKS$', 'GEOMETRY' ,
        MDSYS.SDO_DIM_ARRAY(
          MDSYS.SDO_DIM_ELEMENT('LAENGE', -180, 180, 0.000005),
          MDSYS.SDO_DIM_ELEMENT('BREITE',  -90, 90, 0.000005),
          MDSYS.SDO_DIM_ELEMENT('METER',  0, 1000000, 0.05)), 4326 );
COMMIT;

CREATE INDEX Basis_Links_geom_ix
ON BASIS_LINKS$(geometry)
```

```
INDEXTYPE IS MDSYS.SPATIAL_INDEX;


-- Kilometrierung / Kosten erzeugen:

UPDATE BASIS_LINKS$
SET geometry = SDO_LRS.CONVERT_TO_LRS_GEOM(geometry);

UPDATE BASIS_LINKS$
SET cost = SDO_GEOM.SDO_LENGTH(geometry,0.001,'unit=M');
COMMIT;


-- Routing auf Basis-Netzwerk:

DECLARE
BEGIN
  SDO_NET_MEM.NETWORK_MANAGER.READ_NETWORK('Basis', 'TRUE');
END;
/

CREATE OR REPLACE FUNCTION shortestPath (
  startnode IN NUMBER, destnode IN NUMBER) RETURN SDO_NUMBER_ARRAY IS
  pathId NUMBER;
BEGIN
  SDO_NET_MEM.NETWORK_MANAGER.READ_NETWORK('Basis', 'TRUE');
  pathId := SDO_NET_MEM.NETWORK_MANAGER.SHORTEST_PATH('Basis',startnode,destnode);
  SDO_NET_MEM.NETWORK.ADD_PATH('Basis', pathId);
  RETURN SDO_NET_MEM.PATH.GET_NODE_IDS('Basis', pathId);
END;
/

SELECT shortestPath(725,438) FROM DUAL;

SELECT * FROM TABLE(shortestPath(725,438)) t;
```