7.4 Zugriffsrechte

Benutzeridentifikation

- Benutzername
- Password
- `sqlplus /`: Identifizierung durch UNIX-Account

Zugriffsrechte innerhalb ORACLE

- Zugriffsrechte an ORACLE-Account gekoppelt
- initial vom DBA vergeben

Schemakonzept

- Jedem Benutzer ist sein *Database Schema* zugeordnet, in dem "seine" Objekte liegen.
- Bezeichnung der Tabellen *global* durch
  `<username>.<table>`
  (z.B. `dbis.City`),
- im eigenen Schema nur durch `<table>`.

Systemprivilegien

- berechtigen zu Schemaoperationen
- `CREATE [ANY] TABLE/VIEW/TYPE/INDEX/CLUSTER/TRIGGER/PROCEDURE`:
  Benutzer darf die entsprechenden Schema-Objekte erzeugen,
- `ALTER [ANY] TABLE/TYPE/TRIGGER/PROCEDURE`:
  Benutzer darf die entsprechenden Schema-Objekte verändern,
- `DROP [ANY] TABLE/VIEW/TYPE/INDEX/CLUSTER/TRIGGER/PROCEDURE`:
  Benutzer darf die entsprechenden Schema-Objekte löschen.
- `SELECT/INSERT/UPDATE/DELETE [ANY] TABLE`:
  Benutzer darf in Tabellen Tupel lesen/erzeugen/verändern/entfernen.
- `ANY`: Operation in *jedem* Schema erlaubt,
- ohne `ANY`: Operation nur im eigenen Schema erlaubt

Praktikum:

- `CREATE SESSION, ALTER SESSION, CREATE TABLE, CREATE VIEW, CREATE SYNONYM, CREATE PROCEDURE...`
- Zugriffe und Veränderungen an den eigenen Tabellen nicht explizit aufgeführt (`SELECT TABLE`).

SYSTEMPRIVILEGIEN

```
GRANT <privilege-list>
TO <user-list> | PUBLIC [ WITH ADMIN OPTION ];
```

- `PUBLIC`: jeder erhält das Recht.
- `ADMIN OPTION`: Empfänger darf dieses Recht weiter vergeben.

Rechte entziehen:

```
REVOKE <privilege-list> | ALL
FROM <user-list> | PUBLIC;
```

nur wenn man dieses Recht selbst vergeben hat (im Fall von `ADMIN OPTION` kaskadierend).

Beispiele:

- `GRANT CREATE ANY INDEX, DROP ANY INDEX`
  `TO opti-person WITH ADMIN OPTION;`

  erlaubt `opti-person`, überall Indexe zu erzeugen und zu löschen,
- `GRANT DROP ANY TABLE TO destroyer;`
  `GRANT SELECT ANY TABLE TO supervisor;`
- `REVOKE CREATE TABLE FROM mueller;`

Informationen über Zugriffsrechte im Data Dictionary:

```
SELECT * FROM SESSION_PRIVS;
```


OBJEKTPRIVILEGIEN

berechtigen dazu, Operationen auf existierenden Objekten auszuführen.

- Eigentümer eines Datenbankobjektes
- Niemand sonst darf mit einem solchen Objekt arbeiten, außer
- Eigentümer (oder DBA) erteilt explizit entsprechende Rechte:

  ```
  GRANT <privilege-list> | ALL [(<column-list>)]
  ON <object>
  TO <user-list> | PUBLIC
  [ WITH GRANT OPTION ];
  ```
- `<object>`: `TABLE`, `VIEW`, `PROCEDURE/FUNCTION`, `TYPE`,
- Tabellen und Views: Genauere Einschränkung für `INSERT`, `REFERENCES` und `UPDATE` durch `<column-list>`,
- `<privilege-list>`: `DELETE`, `INSERT`, `SELECT`, `UPDATE` für Tabellen und Views,
  `INDEX`, `ALTER` und `REFERENCES` für Tabellen,
  `EXECUTE` für Prozeduren, Funktionen und `TYPE`n.
- `ALL`: alle Privilegien die man an dem beschriebenen Objekt (ggf. auf der beschriebenen Spalte) hat.
- `GRANT OPTION`: Der Empfänger darf das Recht weitergeben.

OBJEKTPRIVILEGIEN

Rechte entziehen:

```
REVOKE <privilege-list> | ALL
ON <object>
FROM <user-list> | PUBLIC
[CASCADE CONSTRAINTS];
```

- `CASCADE CONSTRAINTS` (bei `REFERENCES`): alle referentiellen Integritätsbedingungen, die auf einem entzogenen `REFERENCES`-Privileg beruhen, fallen weg.
- Berechtigung von mehreren Benutzern erhalten: Fällt mit dem letzten `REVOKE` weg.
- im Fall von `GRANT OPTION` kaskadierend.

Überblick über vergebene/erhaltene Rechte:

```
SELECT * FROM USER_TAB_PRIVS;
```

- Rechte, die man für eigene Tabellen vergeben hat,
- Rechte, die man für fremde Tabellen bekommen hat

```
SELECT * FROM USER_COL_PRIVS;
SELECT * FROM USER_TAB/COL_PRIVS_MADE/RECD;
```

Stichwort: Rollenkonzept

SYNONYME

Schemaobjekt unter einem anderen Namen als ursprünglich abgespeichert ansprechen:

```
CREATE [PUBLIC] SYNONYM <synonym>
  FOR <schema>.<object>;
```

- Ohne `PUBLIC`: Synonym ist nur für den Benutzer definiert.
- `PUBLIC` ist das Synonym systemweit verwendbar. Geht nur mit `CREATE ANY SYNONYM`-Privileg.

Beispiel: Benutzer will oft die Relation “City”, aus dem Schema “dbis” verwenden.

- `SELECT * FROM dbis.City;`
- ```
  CREATE SYNONYM DCity
    FOR dbis.City;

  SELECT * FROM DCity;
  ```

Synonyme löschen: `DROP SYNONYM <synonym>;`
  ```

ZUGRIFFSEINSCHRÄNKUNG ÜBER VIEWS

- `GRANT SELECT` kann nicht auf Spalten eingeschränkt werden.
- Stattdessen: Views verwenden.

```
GRANT SELECT [<column-list>]    -- nicht erlaubt
ON <table>
TO <user-list> | PUBLIC
[ WITH GRANT OPTION ];
```

kann ersetzt werden durch

```
CREATE VIEW <view> AS
  SELECT <column-list>
  FROM <table>;

GRANT SELECT
ON <view>
TO <user-list> | PUBLIC
[ WITH GRANT OPTION ];
```


ZUGRIFFSEINSCHRÄNKUNG ÜBER VIEWS: BEISPIEL

pol ist Besitzer der Relation *Country*, will *Country* ohne Hauptstadt und deren Lage für *geo* les- und schreibbar machen.

View mit Lese- und Schreibrecht für *geo*:

```
CREATE VIEW pubCountry AS
SELECT Name, Code, Population, Area
FROM Country;

GRANT SELECT, INSERT, DELETE, UPDATE
      ON pubCountry TO geo;
```

- Referenzen auf Views müssen separat erlaubt werden:

```
<pol>: GRANT REFERENCES (Code) ON Country TO geo;
<geo>: ... REFERENCES pol.Country(Code);
```

7.5 Optimierung der Datenbank

- möglichst wenige Hintergrundspeicherzugriffe
- Daten soweit wie möglich im Hauptspeicher halten

Datenspeicherung:

- Hintergrundspeicherzugriff effizient steuern
  $\longrightarrow$ Zugriffspfade: Indexe, Hashing
- möglichst viele semantisch zusammengehörende Daten mit *einem* Hintergrundspeicherzugriff holen
  $\longrightarrow$ Clustering

Anfrageoptimierung:

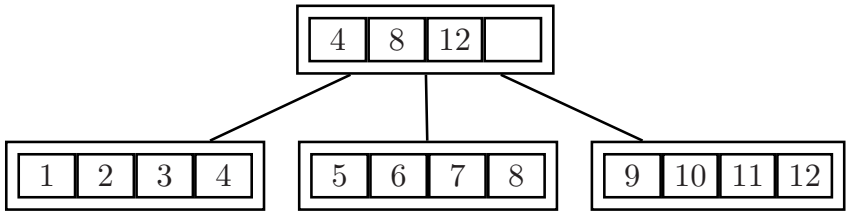
- Datenmengen klein halten
- frühzeitig selektieren
- Systeminterne Optimierung

Algorithmische Optimierung !

ZUGRIFFSPFADE: INDEXE

Zugriff über indizierte Spalte(n) erheblich effizienter.

- Baumstruktur; ORACLE: B*-Mehrweg-Baum,
- B*-Baum: Knoten enthalten *nur* Weg-Information, Verzweigungsgrad hoch, Höhe des Baumes klein.



- Suche durch Schlüsselvergleich: logarithmischer Aufwand.
- Schneller Zugriff (logarithmisch) versus hoher Reorganisationsaufwand ($\rightarrow$ Algorithmentechnik),
- bei sehr vielen Indexen auf einer Tabelle kann es beim Einfügen, Ändern und Löschen von Sätzen zu Performance-Verlusten kommen,
- logisch und physikalisch unabhängig von den Daten der zugrundeliegenden Tabelle,
- keine Auswirkung auf die *Formulierung* einer SQL-Anweisung, nur auf die *interne* Auswertung,
- mehrere Indexe für eine Tabelle möglich.

ZUGRIFFSPFADE: INDEXE

Zugriff über indizierte Spalte(n) erheblich effizienter:

- benötigte Indexknoten aus Hintergrundspeicher holen,
- dann nur ein Zugriff um ein Tupel zu bekommen.

```
SELECT Name, Code FROM Country WHERE Code > 'M';
```

- Ausgabe alphabetisch nach Code geordnet:
  Auf Schlüsselattribut ist automatisch ein Index angelegt und wird verwendet.

```
SELECT Name, Population
FROM Country
WHERE Population > 50000000;
```

- Ausgabe nicht sinnvoll geordnet:
  kein Index vorhanden, linearer Durchlauf (“Scan”).

```
CREATE INDEX CountryPopIndex ON Country (Population);
SELECT Name, Population
FROM Country
WHERE Population > 50000000;
```

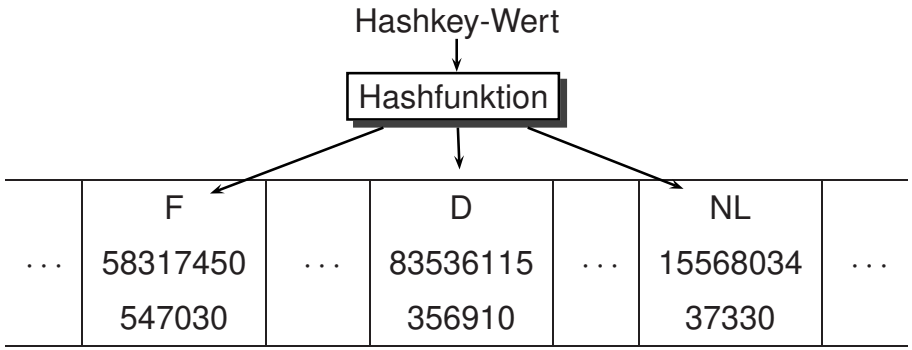
- Ausgabe jetzt nach Population geordnet.
  (Zugriff auf 50000000 über Baum, Rest linear ausgeben)

```
DROP INDEX CountryPopIndex;
```


HASHING

Aufgrund der Werte einer/mehrerer Spalten (*Hashkey*) wird durch eine *Hashfunktion* berechnet, wo das/die entsprechende(n) Tupel zu finden sind.

- Zugriff in *konstanter* Zeit,
- keine Ordnung.
- gezielter Zugriff auf die Daten über ein bestimmtes Land
  Hashkey: Country.Code



In ORACLE ist Hashing nur für *Cluster* implementiert.

CLUSTER

- Zusammenfassung einer Gruppe von Tabellen, die alle eine oder mehrere gemeinsame Spalten (Clusterschlüssel) besitzen, oder
- Gruppierung einer Tabelle nach dem Wert einer bestimmten Spalte (Clusterschlüssel);
- bei einem Hintergrundspeicherzugriff werden semantisch zusammengehörende Daten in den Hauptspeicher geladen.

Vorteile eines Clusters:

- geringere Anzahl an Plattenzugriffen und schnellere Zugriffsgeschwindigkeit
- geringerer Speicherbedarf, da jeder Clusterschlüsselwert nur einmal abgespeichert wird

Nachteile:

- ineffizient bei häufigen Updates der Clusterschlüsselwerte, da dies eine physikalische Reorganisation bewirkt
- schlechtere Performance beim Einfügen in Cluster-Tabellen

CLUSTERING

Sea und geo_Sea mit Clusterschlüssel `Sea.Name`:

`Cl_Sea`		
Mediterranean Sea	`Depth`	
	5121	
	`Province`	`Country`
	Catalonia	E
	Valencia	E
	Murcia	E
	Andalusia	E
	Languedoc-R.	F
	Provence	F
	⋮	⋮
Baltic Sea	`Depth`	
	459	
	`Province`	`Country`
	Schleswig-H.	D
	Mecklenb.-Vorp.	D
	Szczecin	PL
	⋮	⋮

CLUSTERING

City nach (*Province,Country*):

Country	Province			
D	Nordrh.-Westf.	City	Population	...
		Düsseldorf.	572638	...
		Solingen	165973	...
USA	Washington	City	Population	...
		Seattle	524704	...
		Tacoma	179114	...
⋮	⋮	⋮	⋮	⋮

ERZEUGEN EINES CLUSTERS IN ORACLE

Cluster erzeugen und Clusterschlüssel angeben:

```
CREATE CLUSTER <name>(<col> <datatype>-list)
  [INDEX | HASHKEYS <integer> [HASH IS <funktion>]];
```

```
CREATE CLUSTER Cl_Sea (SeaName VARCHAR2(25));
```

Default: *indexed Cluster*, d.h. die Zeilen werden entsprechend dem Clusterschlüsselwert indiziert und geclustert.
Option: `HASH` mit Angabe einer Hashfunktion, nach der geclustert wird.

↪

ERZEUGEN EINES CLUSTERS IN ORACLE

Zuordnung der Tabellen mit `CREATE TABLE` unter Angabe des Clusterschlüssels.

```
CREATE TABLE <table>
  (<col> <datatype>,
   ⋮
   <col> <datatype>)
  CLUSTER <cluster>(<column-list>);
```

```
CREATE TABLE CSea
  (Name   VARCHAR2(25) PRIMARY KEY,
   Depth  NUMBER)
  CLUSTER Cl_Sea (Name);
```

```
CREATE TABLE Cgeo_Sea
  (Province  VARCHAR2(32),
   Country   VARCHAR2(4),
   Sea       VARCHAR2(25))
  CLUSTER Cl_Sea (Sea);
```

Erzeugen des Clusterschlüsselindexes:
(Dies muss *vor* dem ersten DML-Kommando geschehen).

```
CREATE INDEX <name> ON CLUSTER <cluster>;
```

```
CREATE INDEX ClSeaInd ON CLUSTER Cl_Sea;
```