1 SQL

- Der Tennisclub hat sowohl Hobbyspieler als auch Meisterschaftsspieler als Mitglieder.
- Meisterschaftsspieler spielen in Mannschaften gegen andere Clubs.
- Jeder Meisterschaftsspieler hat eine eindeutige Verbandsnummer.
- Der Club hat einige Mannschaften, die an der Meisterschaft teilnehmen.
- Jede Mannschaft hat einen Mannschaftsführer.
- Eine Mannschaft besteht nicht immer aus denselben Spielern. Für jedes Spiel muß daher festgehalten werden, welcher Spieler für welches Team antritt und welches Ergebnis er dabei erreicht hat.
- Ein Spieler kann vom Verband für unfaires Verhalten Strafen bekommen.

Abbildung 1 Angabe ERD Tennisclub

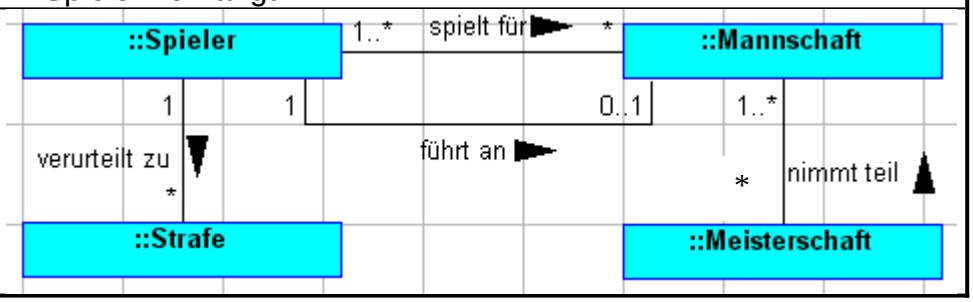
Schritt		Beispiel
1	**Entitäts-mengen bestimmen**	• Spieler • Mannschaft • Strafe
2	**Beziehungen ermitteln**	• 1 Spieler spielt für 0..n Mannschaften; 1 Mannschaft besteht aus 1..n Spielern • 1 Spieler führt 0..1 Mannschaften; 1 Mannschaft wird von 1 Spieler geführt. • 1 Spieler wird verurteilt zu 0..n Strafen; 1 Strafe wird über 1 Spieler verhängt.
3	**ERD erstellen**	

Abbildung 2 Problem -> ERD

Schritt		Beispiel
a	Jede Entitätsmenge wird Relation mit einem **Primärschlüssel**	• Spieler(SpielerId, ...) • Mannschaft(TeamId, ...) • Strafe(StrafId, ...) • Meisterschaft(MeisterId,……)
b	**1:n** - Beziehungen werden Fremdschlüssel	• Strafe(..., *SpielerId*, ...)
c	**m:n** - Beziehungen werden assoziative Tabellen	• Spiel(SpielId, *TeamId*, *SpielerId*, ...) • Teilnahme (TeilnahmeId, *TeamId, MeisterId,*……)
d	Eigenschaften werden **Spalten**	• Spieler(..., Name, VerbandsNr, Geburtdatum, ...) • Mannschaft(..., Name, ...) • Strafe(..., Datum, Betrag, Grund, ...) • Spiel(..., SätzeGewonnen, SätzeVerloren, ...)
e	Jede Beziehung wird zu einem **Fremdschlüssel**	• Mannschaft (..., *KapitänId*, ...)
f	**Normalisierung**	-
g	**Aggregation**	-

Abbildung 3 Überleitung in das relationale Modell

Syntax:
```
CREATE TABLE table_name (
      column_name data_type [default expression] [column integrity rule]
      [, column_name ....,
       ....]
);
```
Beispiel:

Erzeugen der Table PLAYERS
```
create table Players
 (PlayerNo number(4) not null,
  Name varchar2(15),
  date_of_birth date,
  leagueno varchar2(4));
```
Abbildung 4 Erzeugen neuer Tabellen

numerische Datentypen	
NUMBER (P, S)	P … Gesamtanzahl S … Anzahl der Nachkommastellen z.B. NUMBER(6,2) → 0000,00
DECIMAL(P,S)	entspricht NUMBER(P,S)
INT	Ganzzahl; entspricht NUMBER(38,0)
alphanumerische Datentypen	
CHAR(n)	Zeichenfolge mit max. Länge n
VARCHAR2(L)	wie CHAR, jedoch var. Speicherung
LONG	speichert Daten vom Typ VARCHAR; <= 2GB (Variable lang)
Datum	
DATE	es wird Datum und Zeit gespeichert
für Binärdaten, die nicht interpretiert werden	
RAW	uninterpretiert (Sound, Grafik,…)
LONG RAW	uninterpretiert (Sound, Grafik,…) <=2GB
Große Objekte	
BLOB	Große Binärdaten vom Typ RAW <= 4GB
CLOB	Große Zeichendaten <=4GB
identifiziert eindeutig eine Zeile in einer Tabelle	
ROWID	Pseudospalte, "Datensatznummer" jeder Zeile; in allen Tabellen enthalten

Abbildung 5 Datentypen

	Datentyp	**Attribut**	**Nullability**	**Defaulting**
Players	Number (4) Varchar2 (15) Number (4) Varchar2 (4)	PlayerNo Name Year_of_Birth LeagueNo	Not NULL	
Teams	Number (2) Number (4) Varchar2 (6)	TeamNo PlayerNo Division	Not NULL	
Penalties	Number (4) Number (4) Date Number(7,2)	PaymentNo PlayerNo Pen_Date Amount	Not NULL	 aktuelles Dat. 2000,00
Matches	Number (5) Number (2) Number (4) Number (1) Number (1)	MatchNo TeamNo PlayerNo Won Lost	Not NULL	

Abbildung 6 Erzeugen der Tabellen PLAYERS, TEAMS, PENALTIES und MATCHES

```
CREATE TABLE table_name
      [(column_name [column integrity rule]
      [, column_name ....,
       ....])]
      AS SELECT column_name [, column_name ...,...]
          FROM table_name
```

Abbildung 7 Copying Tables

```
DROP TABLE table_name
```

Abbildung 8 Dropping Tables

```
ALTER TABLE table_name MODIFY (
      column_name [data_type] [column integrity rule]
      [, column_name ....,
       ....]
      )
```

Abbildung 9 Altering Tables

```
ALTER TABLE table_name ADD (
      column_name data_type [default expression] [column integrity rule]
      [, column_name ....,
       ....]
      )
```

Abbildung 10 Add Column

```
CREATE [PUBLIC] SYNONYM synonym_name
      FOR table_name
```

Abbildung 11 Synonyms

Numerische Literale:
• Integer
• Decimal
• Floating Point
	`SELECT *`
	`FROM matches`
	`WHERE won = 3`
	`SELECT *`
	`FROM penalties`
	`WHERE amount > 99.9`
	`SELECT *`
	`FROM penalties`
	`WHERE amount > 0.999E2`

Alphanumerische Literale:
• Begrenzt durch Hochkommas
• Darstellung eine Hochkommas in einer Zeichenkette
	`SELECT *`
	`FROM players`
	`WHERE name = ’Baker’`
	`z.B. 'Müller''s Büro'`

Datumsliterale:	`SELECT *`
	`FROM penalties`
	`WHERE pen_date > ’01-Apr-1982’`

Abbildung 12 Literals

LENGTH	Länge einer Zeichenkette `SELECT name, LENGTH(name), FROM players;` Ergebnis: BAKER 5
DECODE	ermöglicht die Vertextung bzw. Umsetzung eines Feldes: decode(<cn>, <strfrom$_1$>, <strto$_1$>, <strfrom$_2$>, <strto$_2$>, …, <strto$_n$>, <str$_{else}$>) ersetzt in Spalte <cn> die Werte <strfrom$_i$> durch die Werte <strto$_i$>; wenn keiner davon zutrifft und <str$_{else}$> angegeben ist, dann dadurch.
SUBSTR	herausschneiden einer beliebigen Zeichenfolge aus einer anderen Zeichenfolge: SUBST(<cn>, <numbBeginPos>, <numbLength>)
INSTR	zum Finden eines Zeichens oder einer Zeichenkette in einer anderen Zeichenkette: SUBSTR(<cn>, <strSearch>, <numbBeginSearch>)
UPPER	wandelt übergebene Zeichenfolge in Großbuchstaben um
LOWER	wandelt übergebene Zeichenfolge in Kleinbuchstaben um

Abbildung 13 Skalarfunktionen

FormatString	Bemerkung
DD, Dy, Day	Tage
MM, Mon, Month	Monat: Mon (3-stellige Kodierung, zB JAN, FEB) Month (in englisch geschriebene Monatsnamen)
YY, YYYY	Jahr (2 oder 4-stellig)
HH, HH12, HH24	Stunden (12 oder 24 Stunden)
MI	Minute
SS	Sekunden

Abbildung 14 Datumsfunktionen

```
SELECT ....
FROM ....
[WHERE ....]
[CONNECT BY ....]
[GROUP BY ....
    [HAVING ....]]
[ORDER BY ....]
```

Abbildung 15 Klauseln eines SELECT Statements

COUNT	Anzahl von Zeilen bzw. Anzahl von Werten (ungleich NULL)
MIN	Minimum
MAX	Maximum
SUM	Summe
AVG	Durchschnitt
STDDEV	Standardabweichung
VARIANCE	Varianz

Abbildung 16 statistische Funktionen

```
expr1 [NOT] BETWEEN expr2 AND expr3
```

Abbildung 17 BETWEEN Operator

```
expr1 [NOT] IN expr2
```

Abbildung 18 IN Operator

```
expr1 [NOT] LIKE expr2
```

Abbildung 19 LIKE *Operator*

```
expr IS [NOT] NULL
```

Abbildung 20 NULL *Operator*

If a is:	Condition	Evaluates to:
10	a IS NULL	FALSE
10	a IS NOT NULL	TRUE
NULL	a IS NULL	TRUE
NULL	a IS NOT NULL	FALSE
10	a = NULL	UNKNOWN
10	a != NULL	UNKNOWN
NULL	a = NULL	UNKNOWN
NULL	a != NULL	UNKNOWN
NULL	a = 10	UNKNOWN
NULL	a != 10	UNKNOWN

Abbildung 21 Beispiele NULL Operator

Ausgabe von Spielernummer und -name derjenigen Spieler, die mindestens eine Strafe erhalten haben:

```
SELECT * FROM players
WHERE EXISTS (SELECT * FROM penalties WHERE
 playerno=players.playerno);
```

Abbildung 22 Lösung

Ausgabe der Spieler mit den 4 höchsten Strafen:

```
select pl.playerno, name, amount
from players pl, penalties pe
where pl.playerno = pe.playerno
and 4 > (select count(*)
         from penalties
         where amount > pe.amount);
```

Abbildung 23 Lösung

Ausgabe der Spieler, die mindestens eine Strafe über 50,00 erhalten haben:

```
select playerno, name
from players p
where exists (select *
              from penalties
              where playerno = p.playerno
              and amount > 50);
```

Abbildung 24 Lösung

- Ausgabe der Spieler, für die jede Strafe über 50,00 war (keine Strafe unter 50,00):

```
select playerno, name
from players p
where not exists (select *
              from penalties
              where playerno = p.playerno
              and amount <= 50);
```

Abbildung 25 Lösung

Allerdings werden dabei auch die Spieler ausgegeben, die überhaupt keine Strafe erhalten haben.
Daher:

```
select playerno, name
from players p
where not exists (select *
              from penalties
              where playerno = p.playerno
              and amount <= 50)
and exists (select *
              from penalties
              where playerno = p.playerno);
```

Abbildung 26 Lösung

Ausgabe sämtlicher Spieler mit ihren Strafen.

```
SELECT name, initials, amount FROM players pl, penalties pe
  WHERE pl.playerno = pe.playerno
UNION
SELECT name, initials, 0 FROM players pl
  WHERE NOT EXISTS
    (SELECT * FROM penalties pe WHERE pe.playerno=pl.playerno)
```

Abbildung 27 Lösung

Ausgabe sämtlicher Spieler mit ihren Strafensummen

```
SELECT name, initials, SUM(amount) FROM players pl, penalties pe
  WHERE pl.playerno = pe.playerno GROUP BY name, initials
UNION
SELECT name, initials, 0 FROM players pl
  WHERE NOT EXISTS
    (SELECT * FROM penalties pe WHERE pe.playerno=pl.playerno)
```

Abbildung 28 Lösung

Syntax:

```
CONNECT BY [PRIOR] condition [START WITH condition]
```

Abbildung 29 Connect by

Folgende Tabelle PARTS:

SUB	SUP	PRICE
P1		130
P2	P1	15
P3	P1	65
P4	P1	20
P9	P1	45
P5	P2	10
P6	P3	10
P7	P3	20
P8	P3	25
P12	P7	10
P10	P9	12
P11	P9	21

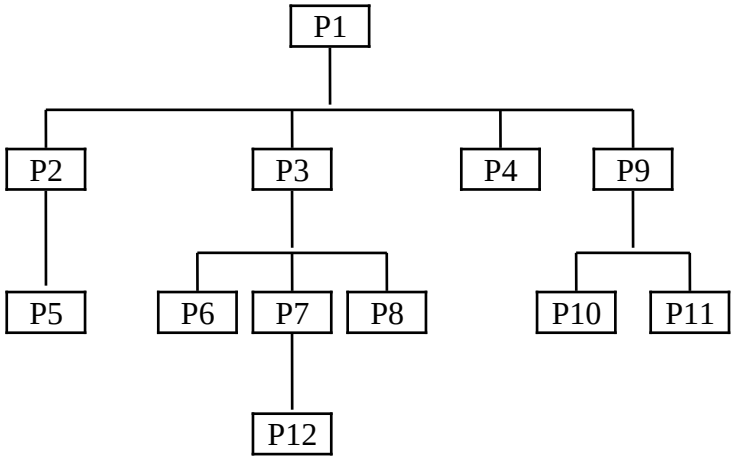


Abbildung 30 Tabelle PARTS

```
INSERT INTO table_name [(col_name1,col_name2,....)]
VALUES (wert1,wert2,....)
```

Abbildung 31 INSERT

Hinweise

- Angabe der Spaltennamen: Spalten nicht notwendig; wenn sie weggelassen werden, müssen allerdings die Werte für alle Spalten in der richtigen Reihenfolge angegeben werden

- NULL Werte: können eingefügt werden, indem
  a) die Spalte nicht belegt wird (unter Angabe des Spaltennamens)

```
INSERT INTO dept (deptno, dname)
  VALUES    (60, 'MIS');
```

  oder

  b) das Schlüsselwort NULL verwendet wird (in beiden Fällen)

```
INSERT INTO dept
  VALUES    (60, 'MIS', NULL);
```

Abbildung 32 INSERT-Hinweis

```
INSERT INTO table_name [(col_name1,col_name2,....)]
SELECT ....
```

Abbildung 33 Masseninsert

Beispiele: Preis von P05 auf ATS 100,- setzen.	
Preis von P05 um 10% erhöhen.	
Alle Preise über ATS 60,- um 10% herabsetzen.	
Alle Preise unter dem Durchschnitt um 20% erhöhen.	

Abbildung 34 Beispiel - update

```
DELETE FROM table_name
[WHERE condition]
```

Abbildung 35 DELETE

- Beginn einer Transaktion:
  1. mit der ersten ausführbaren DML-Anweisung
  2. mit SAVEPOINT

  savepoint updtsal;
  update emp
    set sal=sal*1.1;
  rollback to updtsal;

- Ende einer Transaktion:

1.COMMIT oder ROLLBACK
2.DDL oder DCL-Anweisung wird ausgeführt (implizites bzw. automatisches COMMIT)
3.Bestimmte Fehler, Exit oder Systemabsturz

Abbildung 36 Hinweis - Transaktionen

```
SELECT MAX(teamno)+1 FROM teams;
INSERT INTO team VALUES (…);
```

Abbildung 37 Variante 1 – max

```
SELECT MAX(teamno)+1 FROM team;
INSERT INTO team VALUES (…);
INSERT INTO teamNo VALUES (…);
```

Abbildung 38 Variante 2 – eigene Nummerntabelle

Anlegen einer Tabelle: `seq(Tablename, NextFreeID)`

```
SELECT NextFreeID FROM Seq WHERE tablename = 'TEAMS';
UPDATE seq SET NextFreeID+1 WHERE tablename = 'TEAMS';
INSERT INTO teams VALUES (…);
```

Abbildung 39 Variante 2a - eigene Nummerntabelle für sämtliche Tabellen mit Surrogaten

```
CREATE SEQUENCE seq_name
[START WITH integer]
[INCREMENT BY integer]
[{MAXVALUE integer | NOMAXVALUE}]
[{MINVALUE integer | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{ORDER] | NOORDER}]
[{CACHE integer | NOCACHE}]
```

Abbildung 40 Variante 3 - eigene Sequence

Beispiel:

```
create sequence seq_teamno start with 3;
INSERT INTO teams(teamno, playerno, division)
VALUES (seq_teamno.NEXTVAL, 104, 'first');
```

Abbildung 41 Variante 3 - eigene Sequence (Beispiel)

- Erstellen einer Nummernfolge:

```
CREATE SEQUENCE dept_deptno
   INCREMENT BY 1
   START WITH 91
   MAXVALUE 100
   NOCACHE
   NOCYCLE;
```

- Sequenz verwenden:

```
INSERT INTO dept(deptno, dname, loc)
VALUES (dept_deptno.NEXTVAL, 'MARKETING', 'SAN DIEGO');
```

- Aktuellen Wert anzeigen:

```
SELECT dept_deptno.CURRVAL
FROM dual;
```

Abbildung 42 Sequence (Beispiel)

```
ALTER SEQUENCE seq_name
[INCREMENT BY integer]
[{MAXVALUE integer | NOMAXVALUE}]
[{MINVALUE integer | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{ORDER | NOORDER}]
[{CACHE integer | NOCACHE}]
```

Abbildung 43 Bearbeiten einer Sequenz

```
DROP SEQUENCE seq_name
```

Abbildung 44 Löschen einer Sequenz

```
CREATE TABLE table_name (
     column_name      data_type      [DEFAULT      expression]
[column_constraint]
     [, column_name ...]
     [, table_constraint, ....]
     )
```

bei nachträglichen Erstellen eines Constraint

```
     ALTER TABLE table_name
     ADD (table_constraint)
```

wobei **table_constraint**:

```
     [CONSTRAINT constraint_name]
     constraint_type (column_name1 [,column_name2, ....]))
```

Abbildung 45 Integritätsbedingungen

- column integrity

```
column_name .... [CONSTRAINT constraint_name] PRIMARY KEY
```

- table integrity

```
....
column_name ....,
[CONSTRAINT constraint_name]
PRIMARY KEY(column_name1 [,column_name2,....]),
```

Abbildung 46 Entity Integrity

- column integrity

```
column_name .... [CONSTRAINT constraint_name]
     REFERENCES table_name[(column_name1[,column_name2,....])]
     [ON DELETE CASCADE]
```

- table integrity

```
....
column_name ....,
[CONSTRAINT constraint_name]
FOREIGN KEY (column_name1[,column_name2,....])
REFERENCES table_name [(column_name1[,column_name2,....])]
[ON DELETE CASCADE],
```

Abbildung 47 Referential Integrity

Folgende Tabelle zeigt die DML Operationen, die auf dem Primärschlüssel der PARENT Table (z.B. PLAYERS) und auf dem Fremdschlüssel der CHILD Table (z.B. TEAMS) erlaubt sind. Angenommen wird, dass für den Fremdschlüssel NOT NULL gilt:

DML Statement	**Gegen die PARENT Table**	**Gegen die CHILD Table**
INSERT	Immer OK, falls PK-Wert eindeutig	Nur OK, falls der FK-Wert im PK existiert
UPDATE (Restrict)	*OK, falls keine Zeilen der CHILD Table auf den PK-Wert verweisen*	Nur OK, falls der neue FK-Wert im PK existiert
DELETE (Restrict)	OK, falls keine Zeilen der CHILD Table auf den PK-Wert	Immer OK

	verweisen	
DELETE (Cascade)	Immer OK	Immer OK

Abbildung 48 DML Operationen

column integrity

```
column_name .... [CONSTRAINT constraint_name] CHECK condition
```

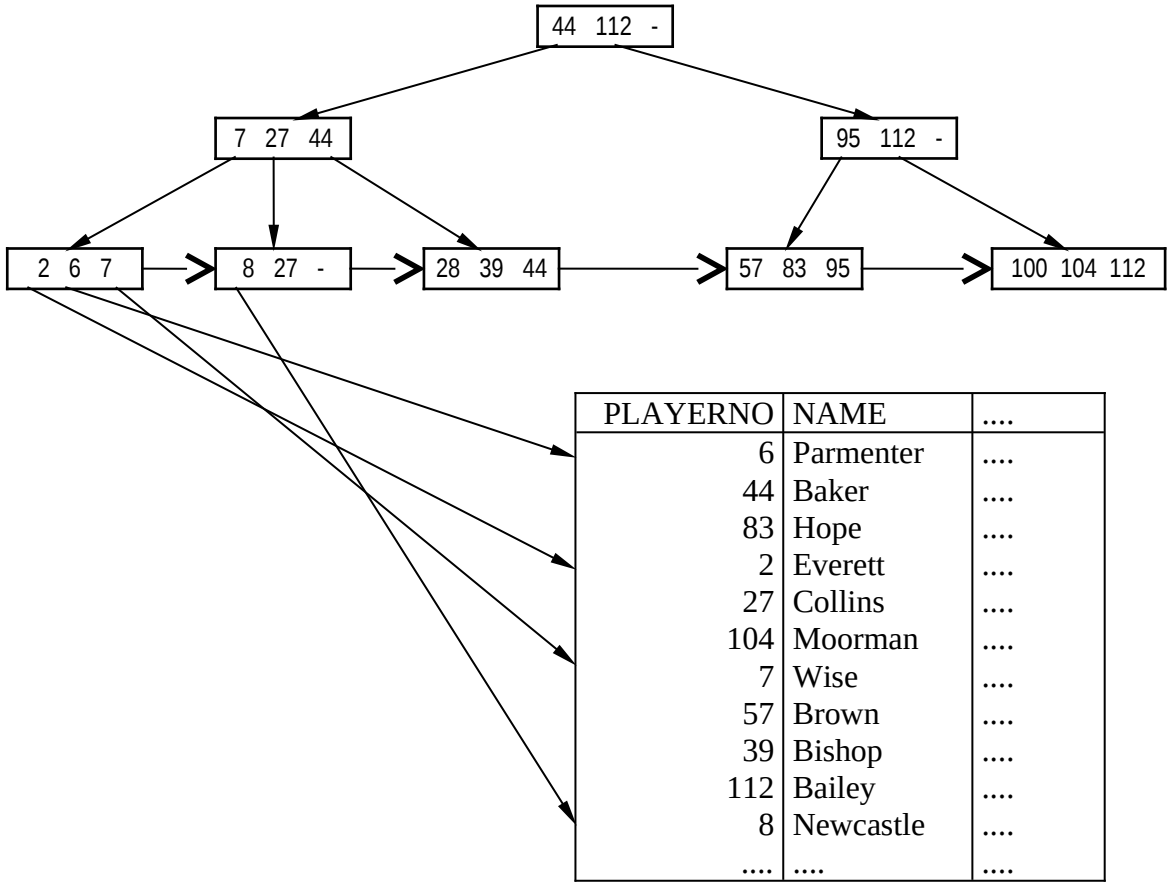
- table integrity

```
....
column_name ....,
[CONSTRAINT constraint_name] CHECK condition,
....
```

Abbildung 49 Check Integrity

```
ALTER TABLE table_name DROP CONSTRAINT constraint_name
```

Abbildung 50 Löschen von Integritätsbedingungen



PLAYERNO	NAME	....
6	Parmenter	....
44	Baker	....
83	Hope	....
2	Everett	....
27	Collins	....
104	Moorman	....
7	Wise	....
57	Brown	....
39	Bishop	....
112	Bailey	....
8	Newcastle	....
....	....	....

Abbildung 51 Indexes

```
CREATE [UNIQUE] INDEX index_name
ON table_name (column_name1[,column_name2,....])
DROP INDEX index_name
```

Abbildung 52 Indexes

Es ist sinnvoll ...	**Es ist nicht sinnvoll ...**
... aus **Integritätsgründen** einen unique index zu erstellen.	... über ein Attribut einen Index zu definieren, das nur **wenige unterschiedliche Werte** enthält.
... auf **Fremdschlüssel** einen Index zu definieren, da die meisten Joins über die Beziehung Primärschlüssel <> Fremdschlüssel laufen.	... eine Abfrage auf <> durch einen Index zu unterstützen.
... über Attribute einen Index zu definieren, wenn nach diesen oft **abgefragt** wird.	... über ein Attribut einen Index zu definieren, das sehr **oft Null** enthält.
... über Attribute einen Index zu definieren, wenn nach diesen oft **sortiert** wird.	

Abbildung 53 Faustregeln für Indexerstellung

- Um den Datenbankzugriff einzuschränken.
- Um komplexe Abfragen einfacher zu machen (Verknüpfung mehrerer Tabellen)
- Um Datenunabhängigkeit zu ermöglichen (z.B. für ad-hoc-Benutzer)
- Um verschiedene Sichten derselben Daten darzustellen.

Abbildung 54 Warum werden Views verwendet

```
CREATE [OR REPLACE] VIEW view_name [(column_name1
      [,column_name2,....])]
AS SELECT ....
[WITH CHECK OPTION [CONSTRAINT constraint_name]]
[WITH READ ONLY]
```

Abbildung 55 Syntax einer View

- Erstellen Sie eine View V_PLAYERS mit PLAYERNO, NAME und die Gesamtsumme der Strafen der einzelnen Spieler (Spalte AMOUNT_TOTAL). Die Spielerdaten sollen auch über die View gesehen werden können, wenn ein Spieler noch keine Strafen erhalten hat.

```
CREATE VIEW v_players AS
SELECT p.playerno,name,sum(amount) AS amount_total
  FROM players p, penalties pe
 WHERE p.playerno = pe.playerno(+)
 GROUP BY p.playerno,name;
```

- Geben Sie sämtliche Spalten der View v_p aus.

```
Select * From v_players;
```

- Geben Sie die Gesamtsumme der Strafen aus.

```
Select Sum(amount_total) From v_players;
```

- Welche Views haben Sie bereits angelegt ?

```
SELECT view_name From user_views;
```

Abbildung 56 Beispiele einer View

Die **Datenbanksicherheit** kann in zwei Kategorien klassifiziert werden:

1. Systemsicherheit
2. Datensicherheit

Systemsicherheit deckt den Zugriff auf und die *Nutzung der Datenbank auf Systemebene* ab (z.B. Benutzername und Passwort) und weist den Benutzern Speicherplatz und Systemoperationen zu (die der Benutzer durchführen darf).
Datensicherheit deckt den Zugriff auf und die *Nutzung der Datenobjekte* ab, sowie die Aktionen, welche der Benutzer mit diesen Objekten durchführen darf.

Abbildung 57 Zwei Kategorien von Datenbanksicherheit

Aus dem Begriff der Datenbanksicherheit leiten sich zwei Arten von Privilegien ab:
Systemprivilegien (system privileges): Zugriff auf die Datenbank erhalten
Objektprivilegien (schema object privileges): Den Inhalt der Datenbankobjekte manipulieren.

Abbildung 58 Arten von Datenbanksicherheit

Ein identifizierter Benutzer hat also nur auf bestimmte Daten Zugriff:

- Daten von Tabellen und Views, die er selbst erstellt hat
- Daten von Tabellen und Views, für die er Zugriffsrechte (z.B. SELECT, UPDATE, ...) erhalten hat

Abbildung 59 Zugriffsbestimmungen

```
GRANT CONNECT
TO    system
IDENTIFIED BY neupwd
```

oder

```
ALTER USER system
IDENTIFIED BY neupwd
```

(DBA Rechte erforderlich)

Abbildung 60 User SYS und SYSTEM

```
CREATE USER user
```

```
IDENTIFIED BY password

GRANT system_privilege [, system_privilege...]
TO user [, user...]
[WITH ADMIN OPTION]
```

Abbildung 61 Anlegen von Usern und Vergabe von Rechten

Systemprivileg	**Operationen. für die die Berechtigung gilt**
CREATE USER	Es dürfen andere Benutzer angelegt werden
DROP USER	Ein anderer Benutzer darf gelöscht werden
DROP ANY TABLE	Eine Tabelle darf in beliebigen Schema gelöscht werden
BACKUP ANY TABLE	Eine beliebige Tabelle in einem beliebigen Schema darf mit der Export Utility gesichert werden.

Abbildung 62 Beispiele für typische Systemprivilegien des DBA

Systemprivileg	**Operationen. für die die Berechtigung gilt**
CREATE SESSION	Verbindung zur Datenbank herstellen
CREATE TABLE	Tabellen im Schema des Benutzers erstellen
CREATE SEQUENCE	Eine Sequenz im Schema des Benutzers erstellen
CREATE VIEW	Eine View im Schema des Benutzers erstellen
CREATE PROCEDURE	Eine Stored Procedure, Funktion oder Package im Schema des Benutzers erstellen.

Abbildung 63 Beispiele für typische Systemprivilegien von Benutzern

```
REVOKE {system_privilege [,system_privilege]|ALL}
FROM   user [, user, ....]
```

Abbildung 64 User Rechte zurücknehmen

```
DROP USER user [CASCADE]
```

Abbildung 65 User entfernen

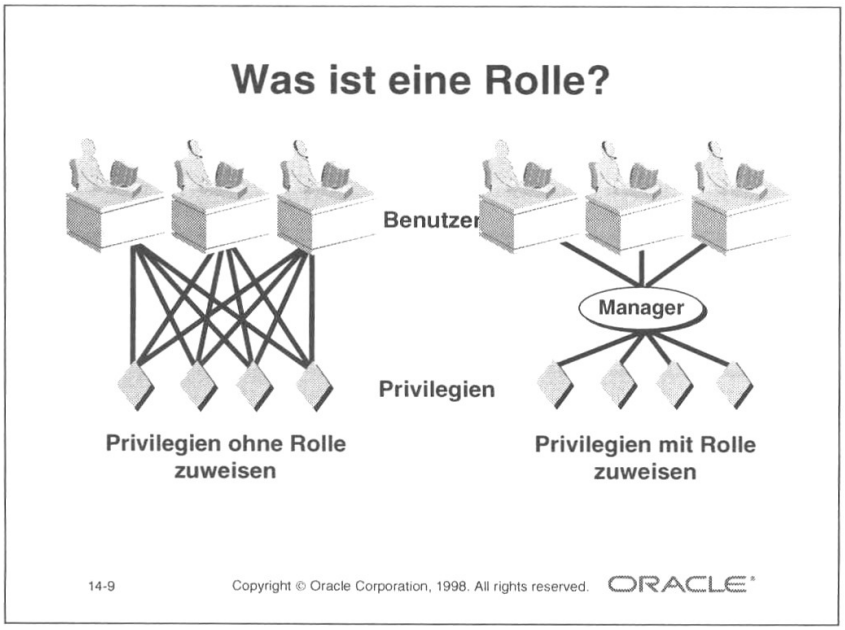


Abbildung 66 Was ist eine Rolle

```
CREATE ROLE role

GRANT role [, role]
TO    user [,user, ....]
[IDENTIFIED BY password [, password, ....]]
```

Abbildung 67 eine Rolle anlegen

Es gibt in ORACLE zwei Arten von User-Accounts

1. User-Accounts, die an das Betriebssystem gebunden sind.
   z.Bsp.: Windows-Login: D3BH08 ergibt einen ORACLE account OPS$D3BH08
2. Gewöhnliche Accounts wie SCOTT/TIGER

Abbildung 68 OPS$user

Objektprivileg	Tabelle	View	Sequenz	Prozedur
ALTER	✓		✓	
DELETE	✓	✓		
EXECUTE				✓
INDEX	✓			
INSERT	✓	✓		
REFERENCES	✓			
SELECT	✓	✓	✓	
UPDATE	✓	✓		

Abbildung 69 Objektprivilegien

```
GRANT  objekt_priv [(column1 [, column2])]
   ON  objekt
   TO  {user|role|PUBLIC} [, user, ....]
[WITH GRANT OPTION]
```

- wobei objekt_priv folgende Werte annehmen kann:

```
 ALTER
 DELETE
 INDEX
 INSERT
 REFERENCES
 SELECT
 UPDATE [(column_name1 [,column_name2,....])]
 ALL
```

Abbildung 70 Objektrechte vergeben

```
REVOKE {privilege [, privilege...]|ALL}
ON     object
FROM   {user [, user, ....]|role|PUBLIC}
[CASCADE CONSTRAINTS]
```

Abbildung 71 Objektrechte zurücknehmen

Beispiel:
- Tabelle ANGESTELLTER (AngNr, AngName, AngJob, AngGehalt)
- Tabelle wurde vom DB-Administrator (User DBMAN) erstellt.
- Alle Angestellten dürfen lesend auf AngNr, AngName und AngJob zugreifen
- Nur der Präsident mit dem Benutzernamen KING darf alle Operationen auf den Daten durchführen.

Folgende Schritte sind durchzuführen:

```
1. GRANT ALL ON ANGESTELLTER TO KING
2. CREATE VIEW ANG AS
   SELECT ANGNR, ANGNAME, ANGJOB
   FROM ANGESTELLTER
3. GRANT SELECT ON ANG TO PUBLIC
```

- KING muss beim Zugriff DBMAN.ANGESTELLTER angeben, alle anderen Benutzer DBMAN.ANG.

- Wird zusätzlich ein Synonym mit

```
CREATE PUBLIC SYNONYM ANGESTELLTER FOR DBMAN.ANG
```

erzeugt, dann kann jeder mit dem Namen ANGESTELLTER zugreifen.

- Will jedoch KING auf ANGGEHALT zugreifen, so muss er DBMAN.ANGESTELLTER angegeben.

Abbildung 72 Zugriffskontrolle mit Views

Data-Dictionary-Tabelle	**Beschreibung**
ROLE_SYS_PRIVS	an Rollen vergebene Systemprivilegien
ROLE_TAB_PRIVS	an Rollen vergebene Tabellenprivilegien
USER_ROLE_PRIVS	Rollen, die für Benutzer zugänglich sind
USER_TAB_PRIVS_MADE	an Benutzerobjekte vergebene Objektprivilegien
USER_TAB_PRIVS_RECD	erhaltene Objektprivilegien
USER_COL_PRIVS_MADE	an Spalten von Benutzerobjekten vergebene Objektprivilegien
USER_COL_PRIVS_RECD	Objektprivilegien, die der Benutzer für bestimmte Spalten erhalten hat.

Abbildung 73 Data-Dictionary-Tabelle