Fachhochschule Nordwestschweiz
Hochschule für Technik

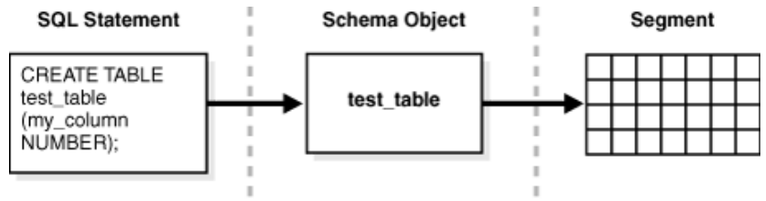
DATENBANK-ARCHITEKTUR FÜR FORTGESCHRITTENE

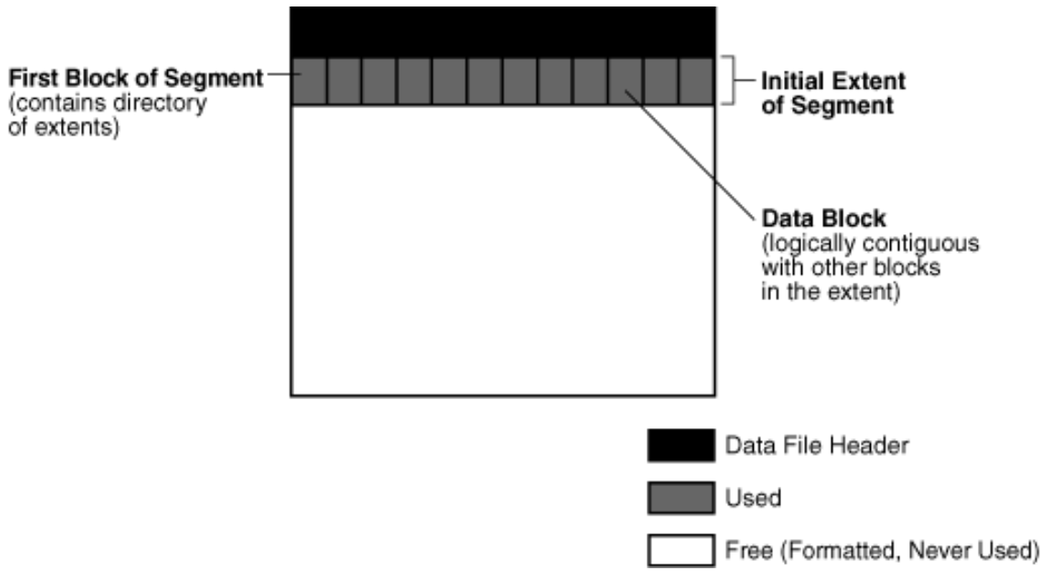
Speicherstruktur von Tabellen

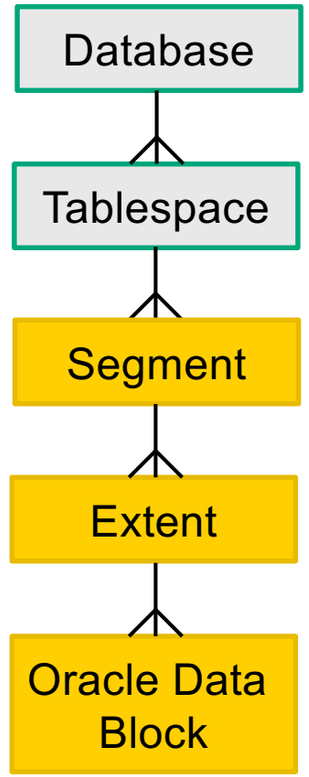
Dani Schnider

Struktur einer Tabelle

- Pro Tabelle wird ein physisches Segment angelegt
- Ein Segment besteht aus einem oder mehreren Extents
- Extents bestehen aus mehreren Data Blocks (typischerweise 8 KB)



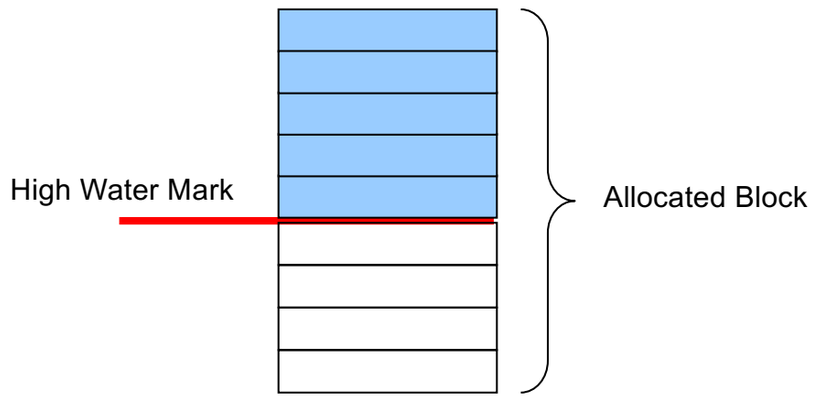




High-Water Mark

- Im ersten Block jedes Segments ist “High-Water Mark”(HWM) gespeichert, d.h. Pointer auf den letzten beschriebenen Block
- Beim Lesen der Tabelle wird nur auf die Blöcke bis zur HWM zugegriffen
- Bei INSERT am Ende der Tabelle wird HWM erhöht
- Bei DELETE bleibt HWM unverändert!

- Zurücksetzen von High-Water Mark nur mit DDL-Befehlen möglich:
  - `TRUNCATE TABLE`
  - `DROP / CREATE TABLE`
  - `ALTER TABLE … SHRINK SPACE`
  - `ALTER TABLE … SHRINK SPACE CASCADE`
  - `ALTER TABLE … MOVE`



Full Table Scan

- Lesen der ganzen Tabelle
- Es werden nur die Blöcke bis zur HWM gelesen
- Full Table Scans werden verwendet
  - Wenn ein grosser Anteil der Tabelle gelesen wird
  - Wenn kein passender Index vorhanden ist
- Beim Lesen wird Multi-block Read Mechanismus verwendet
  - Parameter `db_file_multiblock_read_count`
  - Maximale I/O Grösse: `db_block_size * db_file_multiblock_read_count`

```
SELECT * FROM customers

---------------------------------------
| Id  | Operation          | Name     |
---------------------------------------
|   0 | SELECT STATEMENT   |          |
|   1 |  TABLE ACCESS FULL| CUSTOMERS |
---------------------------------------
```

Table Compression – Grundprinzip

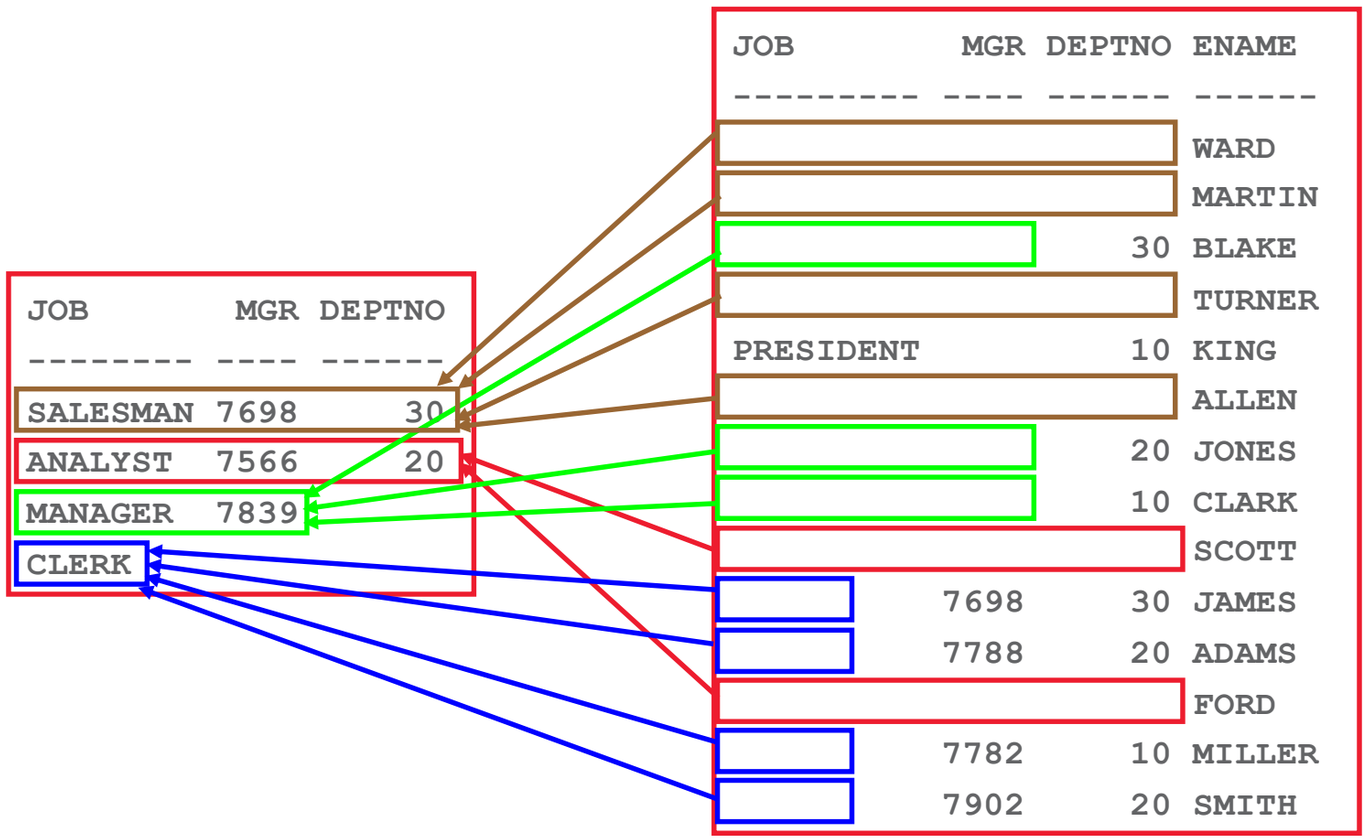


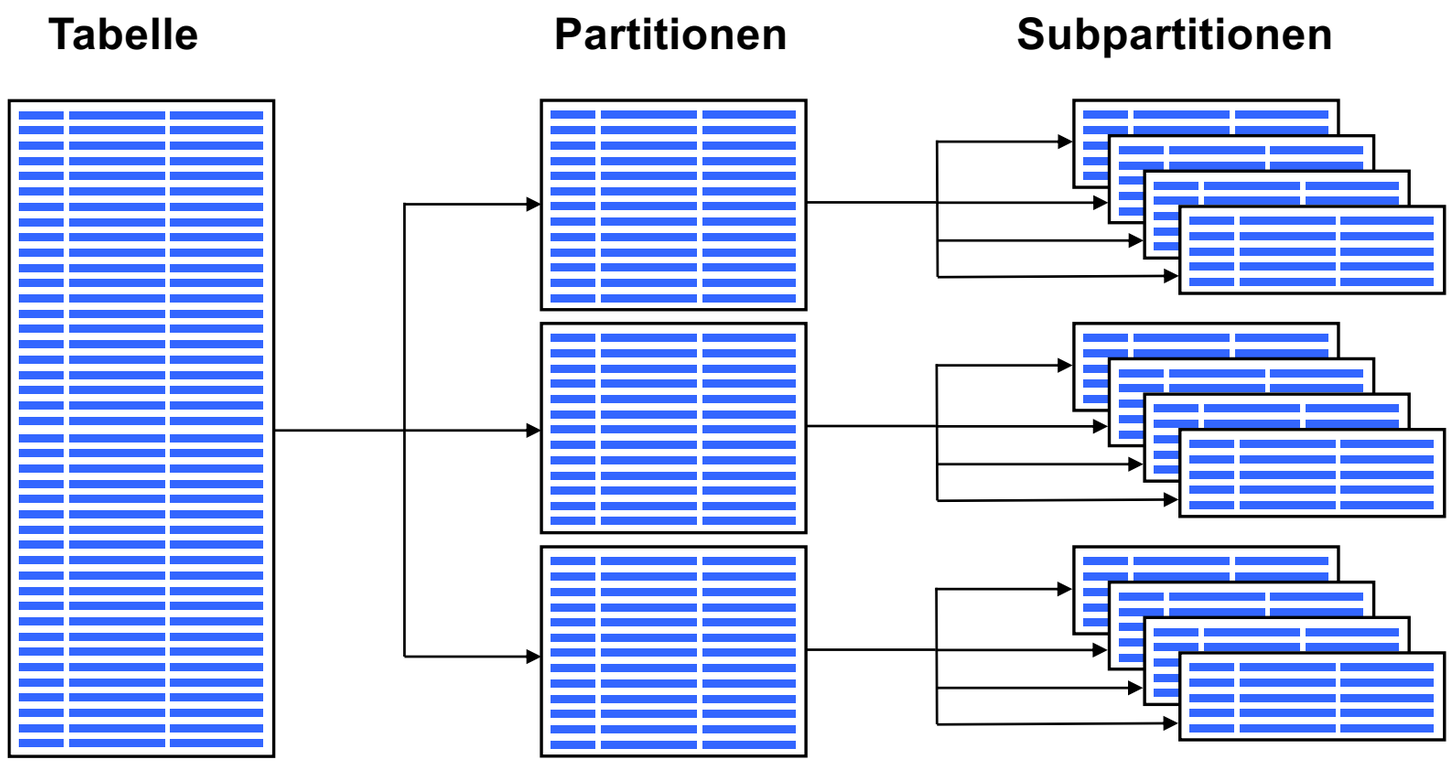
Table Compression

```
CREATE TABLE addresses (…) COMPRESS
```

```
ALTER TABLE addresses MOVE COMPRESS
```

- Wird nur bei Direct-Load INSERT angewendet
  - CREATE TABLE ... AS SELECT ...
  - INSERT /*+ append */ ...
  - Parallel INSERT Statements
- UPDATEs vermeiden!
  - Row Migration, Platzbedarf "explodiert“
  - Schlechte Zugriffsperformance bei Indexzugriffen
- Typische Kompressionsrate: ~ 2:1 bis 4:1
  - Kleiner CPU-Overhead

Grundidee von Partitionierung



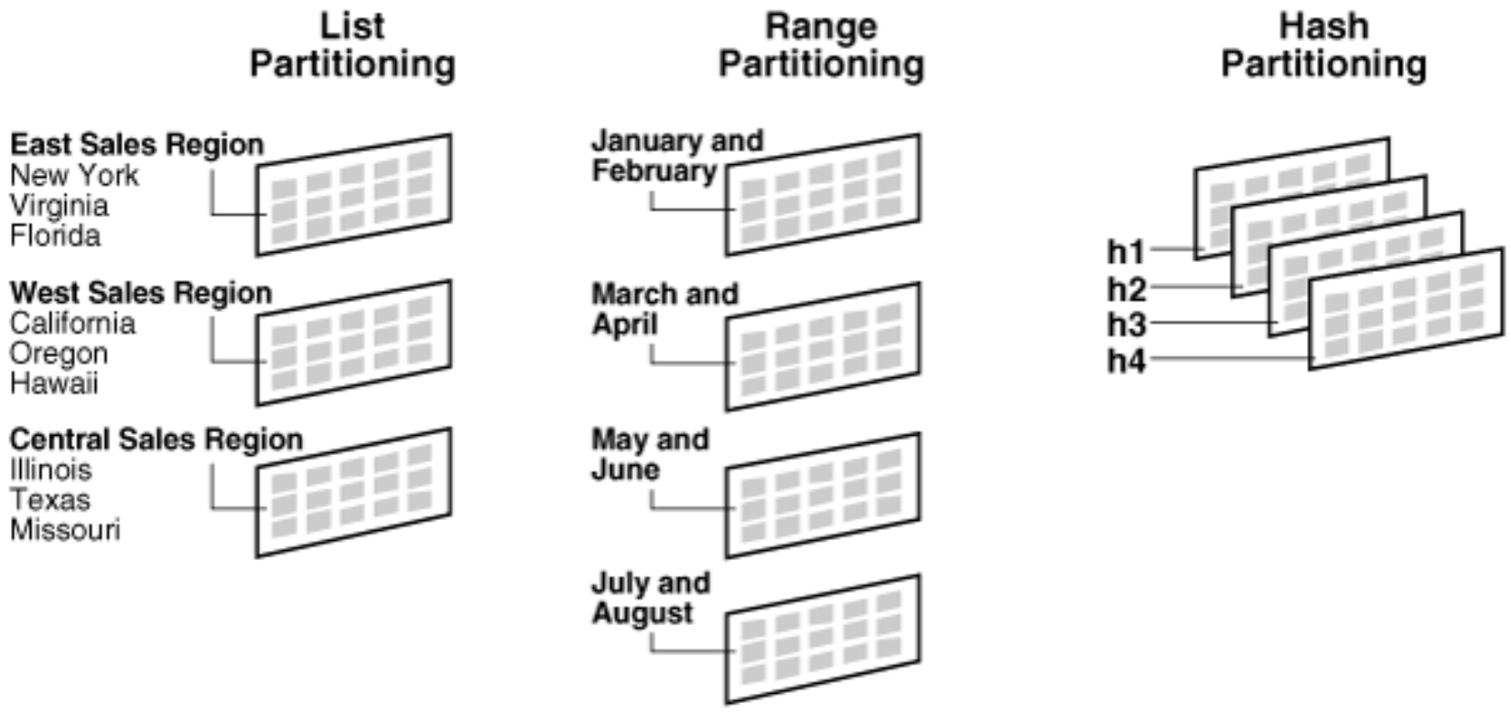
Gründe für Partitionierung

- Abfrageperformance
  - Partition Pruning: Zugriff nur auf relevante Partitionen
  - Partition-wise Join (Full oder Partial Partition-wise Join)
- Rollende Zeitfenster (”Information Lifecycle Management”, ILM)
  - Erstellen neuer Partitionen für periodische Intervalle
  - Löschen von alten Partitionen für nicht mehr benötigte Daten
- Vereinfachte Datenbankadministration
  - Backups nur von aktuellen Partitionen
  - Komprimieren von historischen Partitionen

Praxisbeispiel für effizientes Löschen von alten Daten:

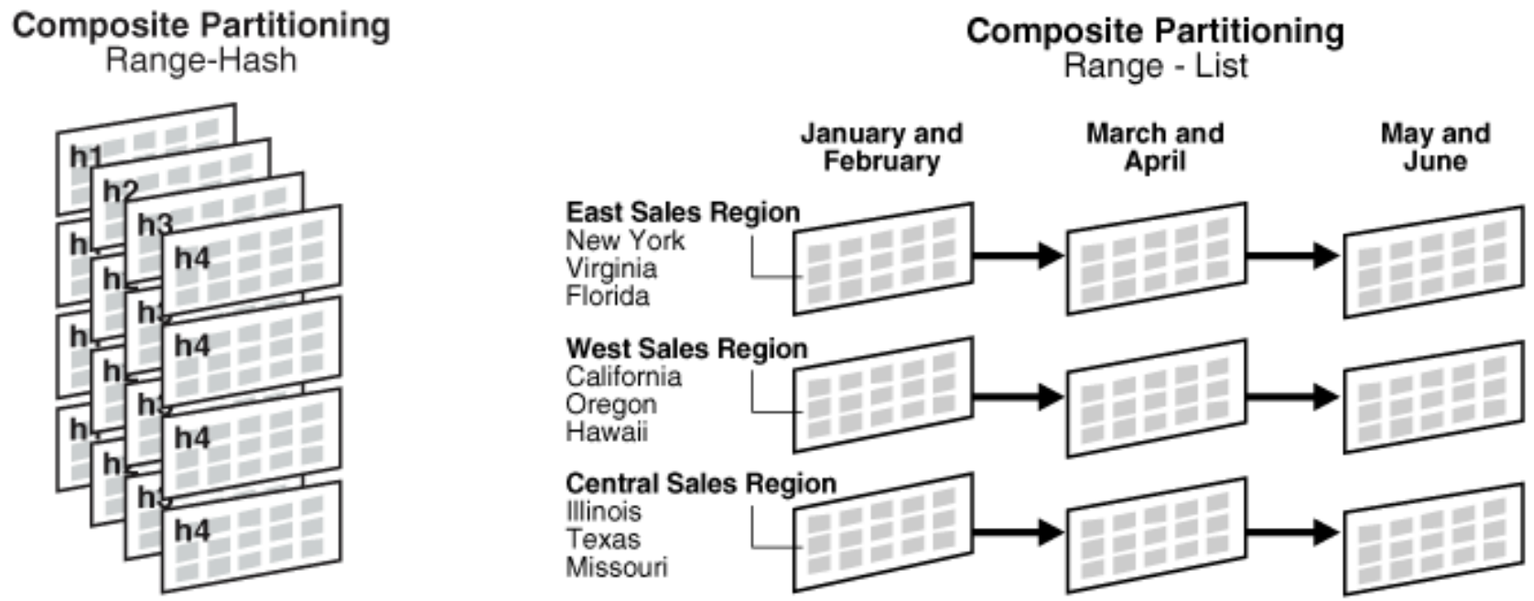
https://danischnider.wordpress.com/2022/07/02/housekeeping-in-oracle-how-to-get-rid-of-old-data/

Partitionierungsmethoden



Quelle: **Oracle Database Documentation, VLDB and Partitioning Guide**
https://docs.oracle.com/en/database/oracle/oracle-database/21/vldbg/partition-concepts.html#GUID-6CE884AF-84A4-4E6A-A3EF-DCCEBCAB2DB2

Composite Partitioning



Quelle: **Oracle Database Documentation, VLDB and Partitioning Guide**
https://docs.oracle.com/en/database/oracle/oracle-database/21/vldbg/partition-concepts.html#GUID-6CE884AF-84A4-4E6A-A3EF-DCCEBCAB2DB2

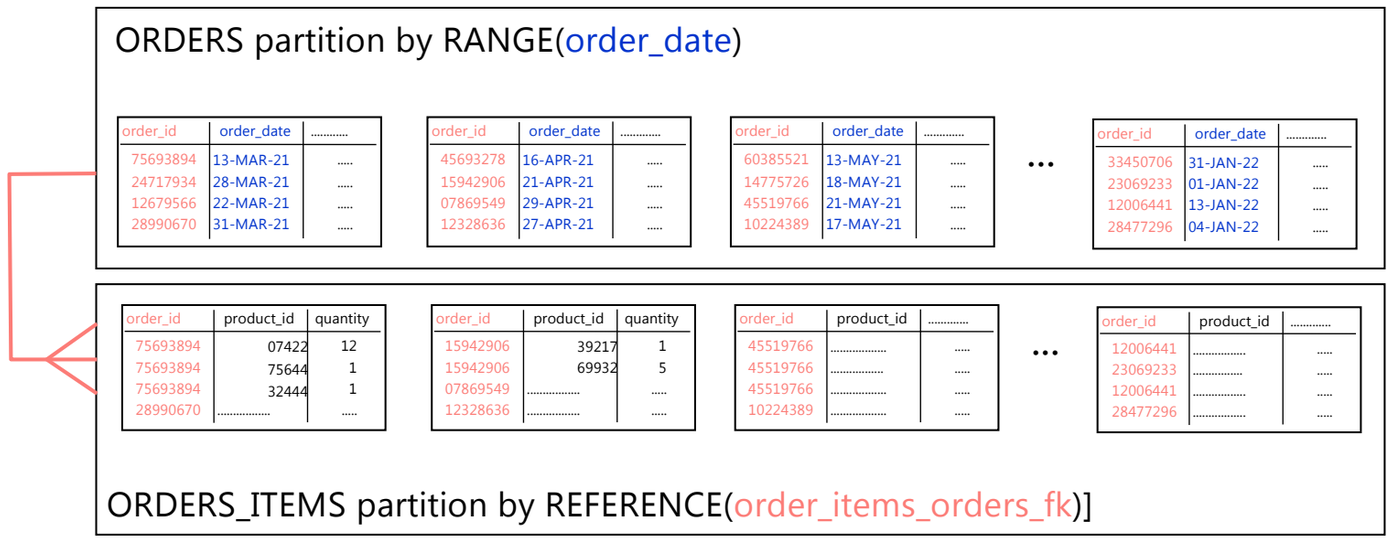
Interval Partitioning

- Erweiterung von RANGE Partitioning ⇨ INTERVAL Partitioning
- Beim Einfügen von neuen Daten wird bei Bedarf neue Partition erstellt

```
CREATE TABLE sales
(time_id   DATE          NOT NULL
,prod_id   NUMBER(6)     NOT NULL
,cust_id   NUMBER(6)     NOT NULL
,quantity  NUMBER(3)     NOT NULL
,amount    NUMBER(10,2)  NOT NULL)
PARTITION BY RANGE (time_id)
INTERVAL(NUMTOYMINTERVAL(1, 'MONTH'))
   (PARTITION p0 VALUES LESS THAN (TO_DATE('01.01.2020','DD.MM.YYYY')))
```

Reference Partitioning

- Parent Table: Partitioniert nach einer der vorgestellten Methoden (z.B. RANGE)
- Child Table: wird gleich partitioniert wie Parent Table
  - Bedingung: Foreign Key Constraint zu Parent Table vorhanden



Praxisbeispiel: Effiziente Speicherverwaltung mit Partitionierung und Compression

1. Set next tablespace to read-write
2. Drop oldest partition
3. Create new partition for next month
4. Compress current partition
5. Set tablespace to read-only

```
ALTER TABLESPACE ts_21
READ ONLY;
```

TS_01 Jan 08	TS_02 Feb 08	TS_03 Mar 08	TS_04 Apr 08	TS_05 Mai 08	TS_06 Jun 08	TS_07 Jul 08	TS_08 Aug 08	TS_09 Sep 08	TS_10 Oct 08	TS_11 Nov 08	TS_12 Dec 08
TS_13 Jan 09	TS_14 Feb 09	TS_15 Mar 09	TS_16 Apr 09	TS_17 Mai 09	TS_18 Jun 09	TS_19 Jul 09	TS_20 Aug 09	TS_21 Sep 09	TS_22 **Oct 09**	TS_23 Nov 06	TS_24 Dec 06
TS_25 Jan 07	TS_26 Feb 07	TS_27 Mar 07	TS_28 Apr 07	TS_29 Mai 07	TS_30 Jun 07	TS_31 Jul 07	TS_32 Aug 07	TS_33 Sep 07	TS_34 Oct 07	TS_35 Nov 07	TS_36 Dec 07

Praxisbeispiel: Effiziente Speicherverwaltung mit Partitionierung und Compression

1. Set next tablespace to read-write
2. Drop oldest partition
3. Create new partition for next month
4. Compress current partition
5. Set tablespace to read-only

```
ALTER TABLESPACE ts_22
READ WRITE;
```

TS_01 Jan 08	TS_02 Feb 08	TS_03 Mar 08	TS_04 Apr 08	TS_05 Mai 08	TS_06 Jun 08	TS_07 Jul 08	TS_08 Aug 08	TS_09 Sep 08	TS_10 Oct 08	TS_11 Nov 08	TS_12 Dec 08
TS_13 Jan 09	TS_14 Feb 09	TS_15 Mar 09	TS_16 Apr 09	TS_17 Mai 09	TS_18 Jun 09	TS_19 Jul 09	TS_20 Aug 09	TS_21 Sep 09	TS_22 **Oct 09**	TS_23 Nov 06	TS_24 Dec 06
TS_25 Jan 07	TS_26 Feb 07	TS_27 Mar 07	TS_28 Apr 07	TS_29 Mai 07	TS_30 Jun 07	TS_31 Jul 07	TS_32 Aug 07	TS_33 Sep 07	TS_34 Oct 07	TS_35 Nov 07	TS_36 Dec 07

Praxisbeispiel: Effiziente Speicherverwaltung mit Partitionierung und Compression

1. Set next tablespace to read-write

```
ALTER TABLESPACE ts_22 READ WRITE;
```

2. Drop oldest partition

```
ALTER TABLE sales DROP PARTITION p_oct_2006;
```

3. Create new partition for next month

```
ALTER TABLE sales
ADD PARTITION p_oct_2009
VALUES LESS THAN
   (TO_DATE('01-NOV-2009', 'DD-MON-YYYY'))
TABLESPACE ts_22;
```

4. Compress current partition

```
ALTER TABLE sales MOVE PARTITION p_sep_2009
COMPRESS;
```

5. Set tablespace to read-only

```
ALTER TABLESPACE ts_21 READ ONLY;
```