A1

```
SELECT * FROM ALL_INDEXES WHERE OWNER = ‚AAZ532‘;
```

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS	COMPRESSION	PREFI
1	AAZ532	PK_GESCHÄFTSPARTNER	NORMAL	AAZ532	GESCHÄFTSPARTNER	TABLE	UNIQUE	DISABLED	
2	AAZ532	PK_LIEFERANTEN	NORMAL	AAZ532	LIEFERANTEN	TABLE	UNIQUE	DISABLED	
3	AAZ532	BONDATEN_PK	NORMAL	AAZ532	BONDATEN	TABLE	UNIQUE	DISABLED	
4	AAZ532	VERKÄUFER_NAME_PK	NORMAL	AAZ532	VERKÄUFER	TABLE	UNIQUE	DISABLED	
5	AAZ532	ARTIKEL_PK	NORMAL	AAZ532	ARTIKEL	TABLE	UNIQUE	DISABLED	
6	AAZ532	ARTIKELGRUPPE_PK	NORMAL	AAZ532	ARTIKELGRUPPE	TABLE	UNIQUE	DISABLED	
7	AAZ532	KUNDEN_PK1	NORMAL	AAZ532	KUNDEN	TABLE	UNIQUE	DISABLED	
8	AAZ532	GESCHÄFT_PK	NORMAL	AAZ532	GESCHÄFTE	TABLE	UNIQUE	DISABLED	
9	AAZ532	LOGS_PL	NORMAL	AAZ532	LOGS	TABLE	UNIQUE	DISABLED	
10	AAZ532	PREISE_PK	NORMAL	AAZ532	PREISE	TABLE	UNIQUE	DISABLED	

A2

```
CREATE TABLE LIEFERANTEN (
  ID NUMBER(10) NOT NULL,
  NAME VARCHAR2(50 BYTE),
  STRAßE VARCHAR2(50 BYTE),
  PLZ VARCHAR2(15 BYTE),
  STADT VARCHAR2(50 BYTE),
  EMAIL VARCHAR2(50 BYTE),
  ANSPRECHPARTNER VARCHAR2(50 BYTE),
  BEWERTUNG VARCHAR2(50 BYTE),
  CONSTRAINT PK_LIEFERANTEN PRIMARY KEY (ID)
);

INSERT INTO LIEFERANTEN
VALUES(1,'MAIER','HAUPTSTRAßE 1','20099','HAMBURG','SUPPORT@MAIER.DE','MARTIN
MAIER','GUT');

INSERT INTO LIEFERANTEN
VALUES(2,'BRAUER KG','HAUPTSTRAßE 2','20095','HAMBURG','EINKAUF@BRAUER-
KG.DE','SABINE GÜNSTIG','ANGENEHM');

INSERT INTO LIEFERANTEN
VALUES(3,'MONTE AG','WEITER WEG 42','42424','BAUERSDORF','VERTRIEB@MONTE-
AG.DE','CARL SCHLEMMER’,'SCHWUNGVOLL');
```

A3

```
CREATE TABLE GESCHÄFTSPARTNER (
  ID NUMBER(10) NOT NULL,
  NAME VARCHAR2(50 BYTE),
  STRAßE VARCHAR2(50 BYTE),
  PLZ VARCHAR2(15 BYTE),
  STADT VARCHAR2(50 BYTE),
  EMAIL VARCHAR2(50 BYTE),
  CONSTRAINT PK_GESCHÄFTSPARTNER PRIMARY KEY (ID)
);


ALTER TABLE KUNDEN
```

```
ADD (GESCHÄFTSPARTNER_ID NUMBER(10));

ALTER TABLE KUNDEN
ADD CONSTRAINT KUNDEN_FK1 FOREIGN KEY (GESCHÄFTSPARTNER_ID) REFERENCES
GESCHÄFTSPARTNER (ID) ENABLE;

ALTER TABLE LIEFERANTEN
ADD (GESCHÄFTSPARTNER_ID NUMBER(10));

ALTER TABLE LIEFERANTEN
ADD CONSTRAINT LIEFERANTEN_FK FOREIGN KEY (GESCHÄFTSPARTNER_ID) REFERENCES
GESCHÄFTSPARTNER (ID) ENABLE;


CREATE OR REPLACE PROCEDURE MIGRATE_GESCHÄFTSPARTENER

IS
  CURSOR C_KUNDEN IS
    SELECT * FROM KUNDEN;

  CURSOR C_LIEFERANTEN IS
    SELECT * FROM LIEFERANTEN;

  KUNDEN_ROW KUNDEN%ROWTYPE;
  LIEFERANTEN_ROW LIEFERANTEN%ROWTYPE;

  ID NUMBER(10);
  NAME VARCHAR2(50 BYTE);
  STRAßE VARCHAR2(50 BYTE);
  PLZ VARCHAR2(50 BYTE);
  STADT VARCHAR2(50 BYTE);
  EMAIL VARCHAR2(50 BYTE);

  G_ID NUMBER(10);

BEGIN
  G_ID := 0;

  OPEN C_KUNDEN;
  LOOP
    FETCH C_KUNDEN INTO KUNDEN_ROW;
    EXIT WHEN C_KUNDEN%NOTFOUND;

    ID := KUNDEN_ROW.ID;
    NAME := KUNDEN_ROW.NAME;
    STRAßE := KUNDEN_ROW.STRAßE;
    PLZ := KUNDEN_ROW.PLZ;
    STADT := KUNDEN_ROW.STADT;
    EMAIL := KUNDEN_ROW.EMAIL;

    G_ID := G_ID + 1;

    INSERT INTO GESCHÄFTSPARTNER VALUES (G_ID, NAME, STRAßE, PLZ, STADT, EMAIL);
    /*DBMS_OUTPUT.PUT_LINE('UPDATE KUNDEN SET GESCHÄFTSPARTNER_ID = ' ||
TO_CHAR(G_ID) || ' WHERE KUNDEN.ID = ' || TO_CHAR(ID));*/
    UPDATE KUNDEN SET GESCHÄFTSPARTNER_ID = G_ID WHERE KUNDEN.ID = ID;COMMIT;
  END LOOP;
  CLOSE C_KUNDEN;

  OPEN C_LIEFERANTEN;
  LOOP
    FETCH C_LIEFERANTEN INTO LIEFERANTEN_ROW;
    EXIT WHEN C_LIEFERANTEN%NOTFOUND;

    ID := LIEFERANTEN_ROW.ID;
```

```
    NAME := LIEFERANTEN_ROW.NAME;
    STRAßE := LIEFERANTEN_ROW.STRAßE;
    PLZ := LIEFERANTEN_ROW.PLZ;
    STADT := LIEFERANTEN_ROW.STADT;
    EMAIL := LIEFERANTEN_ROW.EMAIL;

    G_ID := G_ID + 1;
    INSERT INTO GESCHÄFTSPARTNER VALUES (G_ID, NAME, STRAßE, PLZ, STADT, EMAIL);
    UPDATE LIEFERANTEN SET GESCHÄFTSPARTNER_ID = G_ID WHERE LIEFERANTEN.ID =
ID;COMMIT;
  END LOOP;
  CLOSE C_LIEFERANTEN;

END;
```

```
alter table kunden DROP COLUMN NAME;
alter table kunden DROP COLUMN PLZ;
alter table kunden DROP COLUMN STRAßE;
alter table kunden DROP COLUMN STADT;
alter table kunden DROP COLUMN EMAIL;

alter table lieferanten DROP COLUMN NAME;
alter table lieferanten DROP COLUMN PLZ;
alter table lieferanten DROP COLUMN STRAßE;
alter table lieferanten DROP COLUMN STADT;
alter table lieferanten DROP COLUMN EMAIL;
```

A4

Sequenzen:

```
CREATE SEQUENCE GESCHÄFTSPARTNER_ID_SEQ INCREMENT BY 1 MAXVALUE
9999999999999999999999999999 MINVALUE 15 CACHE 20;

CREATE SEQUENCE KUNDEN_ID_SEQ INCREMENT BY 1 MAXVALUE
9999999999999999999999999999 MINVALUE 50 CACHE 20;

CREATE SEQUENCE LIEFERANTEN_ID_SEQ INCREMENT BY 1 MAXVALUE
9999999999999999999999999999 MINVALUE 15 CACHE 20;
```

Sequenz-Trigger:

```
CREATE TRIGGER KUNDEN_TRG
BEFORE INSERT ON KUNDEN
FOR EACH ROW
BEGIN
  <<COLUMN_SEQUENCES>>
  BEGIN
    IF INSERTING THEN
      SELECT KUNDEN_ID_SEQ.NEXTVAL INTO :NEW.ID FROM SYS.DUAL;
    END IF;
  END COLUMN_SEQUENCES;
END;
/


CREATE TRIGGER LIEFERANTEN_TRG
BEFORE INSERT ON LIEFERANTEN
FOR EACH ROW
BEGIN
```

```
  <<COLUMN_SEQUENCES>>
  BEGIN
    IF INSERTING THEN
      SELECT LIEFERANTEN_ID_SEQ.NEXTVAL INTO :NEW.ID FROM SYS.DUAL;
    END IF;
  END COLUMN_SEQUENCES;
END;
/
```

Prozedur Insert_Kunde:

```
CREATE OR REPLACE PROCEDURE INSERT_KUNDE
(VORNAME IN VARCHAR2, NAME IN VARCHAR2, STRAßE IN VARCHAR2, PLZ IN
VARCHAR2, STADT IN VARCHAR2, EMAIL IN VARCHAR2)

IS
  G_ID NUMBER(10);

BEGIN

  SELECT GESCHÄFTSPARTNER_ID_SEQ.NEXTVAL INTO G_ID FROM SYS.DUAL;
  INSERT INTO GESCHÄFTSPARTNER COLUMN(ID, NAME, STRAßE, PLZ, STADT, EMAIL)
VALUES(G_ID, NAME, STRAßE, PLZ, STADT, EMAIL);
  INSERT INTO KUNDEN COLUMN(VORNAME, GESCHÄFTSPARTNER_ID) VALUES(VORNAME,
G_ID);

END;
```

Prozedur Insert_Kunde:

```
CREATE OR REPLACE PROCEDURE INSERT_LIEFERANT
(ANSPRECHPARTNER IN VARCHAR2, NAME IN VARCHAR2, STRAßE IN VARCHAR2, PLZ IN
VARCHAR2, STADT IN VARCHAR2, EMAIL IN VARCHAR2, BEWERTUNG IN VARCHAR2)

IS
  G_ID NUMBER(10);

BEGIN

  SELECT GESCHÄFTSPARTNER_ID_SEQ.NEXTVAL INTO G_ID FROM SYS.DUAL;
  INSERT INTO GESCHÄFTSPARTNER COLUMN(ID, NAME, STRAßE, PLZ, STADT, EMAIL)
VALUES(G_ID, NAME, STRAßE, PLZ, STADT, EMAIL);
  INSERT INTO LIEFERANTEN COLUMN(ANSPRECHPARTNER, BEWERTUNG,
GESCHÄFTSPARTNER_ID) VALUES(ANSPRECHPARTNER, BEWERTUNG, G_ID);

END;
```

A5

```
create cluster geschäftspartner
(
  g_id number(10)
)
tablespace users;

create index geschäftspartner_cluster
on cluster geschäftspartner
tablespace users;
```

```
create table geschäftspartner_cluster(
  ID NUMBER(10) NOT NULL CONSTRAINT PK_GESCHÄFTSPARTNER_C PRIMARY KEY,
  NAME VARCHAR2(50 BYTE),
  STRAßE VARCHAR2(50 BYTE),
  PLZ VARCHAR2(15 BYTE),
  STADT VARCHAR2(50 BYTE),
  EMAIL VARCHAR2(50 BYTE))
cluster geschäftspartner(ID);

create table kunden_cluster(
  ID NUMBER(10) NOT NULL CONSTRAINT PK_KUNDEN_C PRIMARY KEY,
  VORNAME VARCHAR2(50 BYTE),
  GESCHÄFTSPARTNER_ID number(10) NOT NULL CONSTRAINT
FK_KUNDEN_GESCHÄFTSPARTNER_C REFERENCES geschäftspartner_cluster)
cluster geschäftspartner(ID);

create table lieferanten_cluster(
  ID NUMBER(10) NOT NULL CONSTRAINT PK_LIEFERANTEN_C PRIMARY KEY,
  ANSPRECHPARTNER VARCHAR2(50 BYTE),
  BEWERTUNG VARCHAR2(50 BYTE),
  GESCHÄFTSPARTNER_ID number(10) NOT NULL CONSTRAINT
FK_LIEFERANTEN_GESCHÄFTS_C REFERENCES geschäftspartner_cluster)
cluster geschäftspartner(ID);


insert into geschäftspartner_cluster (select * from geschäftspartner);
insert into kunden_cluster (select * from kunden);
insert into lieferanten_cluster (select * from lieferanten);
```

A6

Auf der oberster Ebene findet ein Hash Join statt. Das heißt basierend auf dem access Prädikat wird ein Hash für die kleinere der beiden Tabellen erstellt und diese dann mit der größeren vermengt.
Unter dieser Ebene gibt es zwei Zweige. Ein Zweig ist ein Table Access (Full), was bedeutet das alles Zeilen der Tabelle unter Berücksichtigung des Filter-Prädikats geladen werden. Ein solcher zugriff ist sehr aufwendig, da hier ggf. alle Daten berücksichtigt werden müssen.
Der zweite Zweig besteht zu nächst aus einem Merge Join, das heißt das hier zwei sortierte Datensätze vermengt werden. Dabei handelt es sich um die Tabelle Geschäftspartner, auf die über den Index zugegriffen wird, und die Tabelle Kunden. Auf Kunden wird ein Full Access mit einem Filter durchgeführt. Anschließend wird die Tabelle mittels eine Access Prädikates sortiert und vermengt.

Auf oberster Ebene findet wieder ein Hash Job statt. In den Zweigen darunter gibt es jedoch unterschiede. So wird auf die Tabelle Bondaten nun nicht mehr mit einem Full Access zugegriffen sondern mittels des erzeugten Index wodurch sich die Kosten verringern.
Im zweiten Zweig bleibt der Menge Join zwischen Geschäftspartnern und Kunden bestehen, wandert aber in der Hierarchie nach unten. Übergeordnet finden nun zunächst zwei Nested Loops statt, Die untergeordnete Loop verknüpft dabei die Kunden/Geschäftspartner Daten mit den Index der Bondaten, die übergeordnete Loop kann nun mittels der indexdaten weitere daten zwischen

Kunden/Geschäftspartner und Bondaten vermengen. Zuletzt werden die Ergebnisse der beiden Zweige vermengt wobei die Reduzierung auf das eigentliche Datum stattfindet. Die gesamt Kosten fallen dabei geringer aus als bei dem Versuch ohne Index. Vermutlich durch die geringe Tabellen große ist der Unterschied jedoch sehr klein.